A Simple Start To jQuery, JavaScript, and HTML5

for Beginners Written by a Software Engineer

SCOTT SANDERSON

A Simple Start to jQuery, JavaScript, and Html5 for Beginners

Written by a Software Engineer

By

Scott Sanderson

Table of Contents

Chapter 1: Introduction

If there is one application development framework that can be termed as comprehensive, then HTML5 wins the bid hands down. Although, the specification for HTML5 is not yet complete, most modern web browsers support the popular features on device, which range from a Smartphone to a desktop. What that means is that you just have to write one application and it will run on your devices without any interoperability issues.

There is no doubt about the fact that HTML5 is the future of web application development and if you wish to remain in the league, you need to think futuristically and equip yourself to deal the technological challenges that the future is about to throw at you. The scope of HTML5 is evident from the fact that most of the major players of the industry and setting their eyes on this technology and giving in full support to the same.

If the multi-faceted functionality and high-on features characteristics of HTML5 intrigue you and you want to start writing your own applications right away, but you do not know how and where to begin, then this book is just for you. This book covers everything that you shall require to create working applications with the help of HTML, JavaScript/JQuery and CSS. However, it is not a reference guide. We hope to give you practical knowledge so that you can get to development as quickly as possible.

This book is a perfect start-up guide and covers all the basic facets of HTML5, JavaScript and CSS development. It covers everything from the very basics to all that you shall require in your tryst with this framework. The first three chapters introduce you to these three technologies, giving you some ground to start with.

Chapter 2: Basics of HTML5

HTML (Hyper Text Markup Language) is a language used for creating web pages. In fact,

this language has been in use since the first webpage was made. However, the functionality has evolved as newer and better versions of the language were introduced. The language is known to have originated from SGML (Standard Generalized Markup Language), which was earlier used for document publishing. HTML has inherited the concept of formatting features and their syntax from SGML.

One of the most interesting and beneficial facet of HTML usage, as far as browsers are concerned, is that browsers support both backward as well as forward compatibility. While backward compatibility is usually easy to achieve, forward compatibility is tricky as the problem domain, in this case, is infinitely large. However, in order to implement this, browsers were designed to ignore tags that it did not recognize.

For years, HTML remained all that people wanted. However, with time, people felt the need for more, which was catalyzed by the presence of another technology called XML (eXtensible Markup Language). Although, XML shares a lot of similarities with HTML, there exist many fundamental differences between the two. Firstly, XML requires tag matching in the sense that for every starting tag, a closing tag must inevitably exist. Besides this, XML allow you to create your own tags as it does not possess a fixed set of tags like HTML.

The tags used in XML are meta-tags or tags that describe the data that is included between the starting and closing tag. In order to ensure the validity of the XML document, a technology called XSD (XML Schema Definition) is used. However, this technology cannot be used for validating HTML documents because HTML documents lack a well-defined structure.

The W3C (World Wide Web Consortium) introduced XHTML as an attempt to fix the flaws of HTML. According to the XHTML specification, HTML documents were forced to adhere to the format specifications used for XML. Therefore, this allowed the use of

XSD tools for validation of HTML documents. Although, the integration of XML in the framework fixed some issues, some issues continued to crop up. One of the staggering issues of the modern times was the growing need for integration of multimedia. While CSS did perform formatting of some level, it was becoming inadequate for the growing demands of users.

In order to provide support for interactivity and animated visuals, a programmable support called JavaScript was added to this ensemble. However, initial versions of this support were difficult for programmers to understand and slow on execution time incurred. This led to the introduction of plug-ins like Flash to get the attention that it did. These plugins did what was expected of them, but the browser-multimedia integration was still loose in nature.

HTML5 is not an evolved form of XHTML. On the contrary, HTML5 can be better described as the reinvented form of HTML 4.01 and how HTML, CSS and JavaScript can be used together to solve the growing needs of the user base.

Semantic Markup

The fundamental feature of HTML5 is that it stresses on separation of behaviour, presentation and structure. The semantic markup of a website development specifies the structure of the document. In other words, it specifies the meaning of tags and what they will do for you. On the other hand, behaviour and presentation are governed by CSS and JavaScript respectively.

HTML5 Elements

In HTML, an element is simply a statement that contains a beginning tag, content and a closing tag. Therefore, when you write,

This is my world!

In this example, the div elements includes everything from to . therefore, the tag is also a part of the div element.

It is important to state here that HTML5 is not case sensitive. Therefore, regardless of whether you write <B> or for the bold tag, the browser will consider the two same. However, the use of lower case for tags is recommended by the W3C standards.

Working with Elements in HTML5

HTML5 defines more than a 100 elements. These elements, with their definitions are provided in Appendix.

How to add attributes to elements?

Additional data can be added to the begin tag in the form of attributes. An attribute can be generally represented as, name=”value”. The value for the attribute name is enclosed within quotes. There is no restriction on the number of attributes that can be added to the begin tag. For these attributes, the attribute has a name, but it does not have a value.

Example:

Here, id and class are attributes where id uniquely identifies the element and class specifies the CSS style to which this div belongs.

Boolean Attributes

Several types of attributes can be used. One of the most commonly used types is Boolean attribute. The Boolean attribute can take a value from the allowable set of values. Some possible values include:

- Checked
- Disabled
- Selected
- Readonly

There are two ways to indicate the value of a Boolean attribute.

<input type=“checkbox” name=“vegetable” value=“Broccoli” checked=” />

<input type=“checkbox” name=“vegetable” value=“Broccoli” checked=‘checked’ />

In the first case, the value is not given and is assumed. Although, the latter seems like a redundant form, it is the more preferred form, particularly if you are using jQuery.

Global Attribute Reference

There is a set of named attributes available in HTML5, which can be used with any element. Examples of these attributes include accesskey, spellcheck and draggable, in addition to several others.

Void Elements

Some elements do not have to contain content under any circumstance. These elements include <link>,
 and <area>, in addition to many others.

Self-closing Tags

If you are using an element that does not contain any content, then you can use a self closing tag. An example of this tag is
. However, please note that other tags like have to be written as even if they do not have any content.

How to Add Expando Attributes

Any attribute that you as the author define is known as expando attribute. You can give this custom attribute a name and assign a value to the same as and when required.

How to Add Comments

Comments can be added using a ! and two hyphens (-). The syntax for adding comments is as follows:

<!—text —>

You can also add conditional comments using the following syntax:

<!—[if lte IE 7]> <html class=“no-js ie6” lang=“en”> <![endif]—>

This comment determines if the browser being used is an earlier version released earlier than IE7.

How to Create HTML Document

An HTML document can simply be described as a frame that contains metadata, content and a structure.

The HTML document must start with the tag <!DOCTYPE html>. In HTML5, the declaration is an indication to the browser that it should work in no-quirks mode or HTML5 compliant mode.

The next tag should be <html>. This element includes within itself the elements <head> and <body>. The <head> element can contain the metadata for the HTML document, which is declared and defined using the <meta> tag. The <head> element also contains the <title> element, which serves the following purposes:

- Used by search engines
- This content is displayed in the browser toolbar
- Gives a default name to the page.

The <body> element includes any other information that is to be displayed on the HTML document.

A sample HTML document is given below:

<!DOCTYPE html>

<html>

<head>

<meta charset=“utf-8” />

<title>TITLE OF DOC</title>

</head>

<body>

CONTENT

</body>

</html>

Note:

1. Special character like ‘>’ or ‘<’, which are a part of the HTML syntax, can be used on the HTML doc using their name or number. The syntax for their usage is:

   &entity_name or &entity_number

   The appendix contains the table of entity names and numbers to be used.

2. White space created using tabs, line breaks and spaces is normalized by HTML into a single space. Therefore, if you want to use multiple spaces, you must use the non-breaking space character. For example, if you want to display 10 mph such that 10 and mph are not separated by a newline, you can write 10 mph.

How to Embed Content

Now that you know how to create HTML documents with your content, you may want to

embed content into the document from other sources. This content may be text from another HTML document or flash applications.

Inline Frames

Inline frames are used to embed HTML documents inline with the content of the current HTML document. Therefore, in a way, this element creates nested browsers as multiple web page are loaded in the same window. These are implemented using the <iframe> element. Nested browsing contexts can be navigated using:

- window.parent – This WindowProxy object represents the parent browsing context.
- window.top – This WindowProxy object represents the top-level browsing context
- window.frameElement – This represents the browsing context container and if there is no context container, it returns null.

The syntax of the <iframe> element is as follows:

<iframe src=“name”></iframe>

Here the name of the attribute defines the name of the browsing context. This name can be any string like ‘sample.html’. However, the strong should not start with an underscore as names starting with underscore are reserved for special key names like _blank.

Sandboxing

Sandboxing is used for avoiding the introduction of pop-ups or any other malware in your HTML document. In order to implement this, the attribute ‘sandbox’ is used. You can use this attribute in the following manner:

<iframe sandbox=”keywords” src=“name”>

</iframe>

When you use sandboxing, several restrictions are imposed on the called context. These restrictions include disabling forms, scripts, plugins and any frame other than itself. You

can force the system to override these restrictions using keywords like:

- allow-same-origin
- allow-scripts
- allow-forms
- allow-top-navigation

Multiple keywords can be used by separating them using a space.

Seamless Embedding

The seamless attribute can be used in the <iframe> element for embedding of content in a manner than this content appears to be part of the main content. This attribute can be used in the following manner:

<iframe seamless=”” src=“name”></iframe>

<iframe seamless src=“name”></iframe>

<iframe seamless=“seamless” src=“name”></iframe>

This attribute may not be supported by many browsers. Therefore, you can use CSS to achieve a similar effect.

Hyperlinks

Hyperlinks can be implemented using the <a> element. This element can be used to link both external (a different HTML document) as well as internal (another location in the same HTML document) links.

All links are underlined and depending upon the nature of the link, the colour of the link changes.

- Blue - Unvisited link
- Purple - Visited link

- Red - Active link

The first attribute of the <a> element is href, which is given the value of the URL.

Syntax for external links:

Text

Syntax for internal links:

Text

The id here is the id of the tag to which you want the link to jump onto. However, if you use only the # value, the link jumps to the top of the page.

The other attribute used with the <a> element is target, which allows you to control the behaviour of the link. For instance, if you want the link to open in another window, then this requirement can be specified using this attribute. The following can be used:

- _blank

  This opens the link in a new browser window.

- _parent

  This opens the link in the parent window.

- _self

  This opens the link in the current window and is the default setting.

- _top

  This opens the link in the topmost frame of the current window.

- <iframe_name>

  This opens the link in the <*iframe*> element of the specified name. This is generally used in menus.

Hyperlinks can also be used to send emails using the following syntax:

Text

When the user clicks on the link, an email will be sent to the specified email address.

Embedding Images

The <img> element is used for adding images to the HTML document. The <img> tag is a void element and does not require a closing tag.

The required tag for this element is *src*, which specifies the absolute or relative address at which the image is stored. Another attribute than can be used with the <img> tag is *target*, which is used to specify the text that must be displayed in case the image is not available.

Syntax:

<img src=”address” target=”alternative text”>

It is important to note that you only give references to images and they are not embedded into the HTML document, in actuality. The allowed image formats are jpeg, svg, png and gif.

How to Create Image Map

The <map> element can be used to create a clickable image map. In order to create a link between the map and image, you must set a name of the map and use this name in the usemap attribute of the img tag.

The area of the map element is defined using <area> element. This is a self-closing tag that is used to define the area of the map. The area of the map can be set using the attributes shape, href, alt and coords.

The shape attribute can be give the values poly, circle, rect or default. The default value sets the area as the size of the image. The href and alt attributes are used in the same

manner as they are used in the <a> element. Lastly, the coords attribute are defined according to the shape chosen in the following manner:

- poly – x_1, y_1, x_2, y_2, … , x_n, y_n
- circle – x, y, radius
- rect - x_1, y_1, x_2, y_2

For the polygon, the starting and ending coordinates should be the same. In case a discrepancy in this regard exists, a closing value is added.

Example:

<img src =“worldmap.gif” width=“145” height=“126”

alt=“World Map” usemap =”#country” />

<map name=“country”>

<area shape=“circle” coords=“105,50,30”

href=“China.html” alt=“China” />

<area shape=“default” href=“InvalidCountry.html” alt=“Please Try Again” />

</map>

Embedding Plug-ins

Plugins can likewise be embedded into HTML documents using <embed> and <object> elements. Although, both these elements perform similar tasks, they were created by different browsers. As a result, they coexist in HTML5. While the <embed> element provides ease of use, the <object> element provides more functionality to the user. Syntax for these two elements is given below:

The <embed> tag

<embed src=“Flash.swf”> </embed>

The src attribute specifies the url or address of the file to be embedded. Other attributes that are taken by the <embed> element includes:

- type - used to specify the MIME type of the content
- height – used to specify the content height in pixels
- width – used to specify the content width in pixels

As mentioned previously, some browsers may not support <embed> element. Therefore, if you are using it in your document, you must add fallback content specification. For instance, if you are embedding a flash file, you must redirect the user to the download link of flash player if the browser does not support it already.

You can do this in the following manner:

<embed src=“Flash.swf” >

<img src=”address of the image for download flash player”

alt=“Download Adobe Flash player” />

</embed>

The <object> tag

The <*object*> tag allows you to embed a variety of multimedia files like video, audio, PDF, applets and Flash. The element accepts the following attributes:

- type – used to specify the MIME type of data
- form – used to specify the form id or ids of the object
- data – used to specify the URL of the resources that the object uses

- usemap – used to specify the name of a client-side image map that the object uses
- name – used to specify the object name
- height – used to specify the height of the object in pixels
- width – used to specify the width of the object in pixels

Of all the attributes mentioned above, it is necessary to mention either the data or type attribute.

Data can be passed into the <object> element using the <param> tag, which is a combination of name and value attributes. Multiple parameters can be declared using this tag.

<object data="file.wav">

<param name="autoplay" value="false" />

</object>

Note:

1. The <object> element must always be used inside the <body> element.
2. HTML5 supports only the attributes listed above and global attributes for the <object> element.

CHAPTER 3: BASICS OF JAVASCRIPT

Interaction is an important facet of any website. In order to connect with the audience in a better way, it is vital to add behaviour to the website. This can be as simple as buttons or as complex as animations. These tasks can be added using JavaScript, which is a web and programming language. This chapter introduces JavaScript and shall help you get started with JavaScript on an immediate basis.

Background

JavaScript is the preferred programming language for client side scripting. Contrary to

popular belief, JavaScript is in no way related to Java. In fact, it finds resemblance to ECMAScript. Besides this, the only common thing between this programming language and other programming languages like C and C++ is that it uses curly braces. The international standard for JavaScript is given in ISO/IEC 16262 and ECMA-262 specification.

One of the most important features of this programming language is that it is untyped. In other words, specifying the type of a variable is not necessary for using it. For example, if you have assigned a string to a variable, you can later assign an integer to the same variable. Variables are declared using the keyword *var*.

Data and Expressions

Any program accesses, manipulates and represents data to the user. Data is available in different types and forms. This data can be easily decomposed into values. In JavaScript, data may be represented as a primitive value, object or function.

The data representation at the lowest level is known as primitive data type and includes null, undefined, number, string and Boolean.

Several built-in objects are defined in JavaScript. These entail the Object object, global object, Array object, Function object, Boolean object, String object, Math object, Number object, the RegExp object, Date object, Error object and JSON object.

Any object that is callable is called a function. It may also be referred to as a method if the function is associated with an object.

Data is produced by using expressions, which is a name given to any code that generates a value. It may be assigned a value directly or the value may be computed by substituting and computing an expression composed of operands and operators. It is important to note that an operand can be another expression as well.

Number Data Type

The number data type is a primitive data type and it is internally stored as a floating point number, which is a 64 bit, double precision number. This 64 bit field stores the sign, exponent and fraction. While the leftmost bit is reserved for sign, the bits 0 to 51 are reserved for storing the fraction and bits 52-62 are used for the exponent.

Because of memory limitation on the system, 253 is the highest integer that can be stored. It is important to note that integer calculations generate a precise value. However, fractional data calculation may give imprecise results. Therefore, these values have to be truncated.

In addition to numbers and strings, JavaScript also supports the use of the following special characters.

- undefined specifies that the value has not been assigned yet.
- *NaN stands for* 'Not a Number'
- -Infinity any number that is less than -1.7976931348623157E + 10308 is denoted by Infinity.
- Infinity - any number that exceeds the value 1.7976931348623157E + 10308 is denoted by Infinity.

String Data Type

A string can simply be described as a collection of characters. Whenever you declare character(s) within quotes, the system interprets as a string. Sample strings include:

'Hello World!'

"Hello World!"

However, if you want to include quotes as characters in the string, then you must add '\' before the character. Sample string:

‘You\’re Welcome’

“You\’re Welcome”

JavaScript also supports other escape sequences like \t for tab and \n for newline.

Boolean Data Type

The Boolean data type is a binary data type and return either true or false. These operators are commonly used to indicate results of comparisons. For example,

10 > 5 will give ‘true’ while 5 > 10 will give ‘false’.

Operations on Number data Type

In order to perform calculations, operators are used. JavaScript supports all the basic operators and the operator precedence is as follows:

- Addition and subtraction have the same precedence.
- Multiplication and division have the same precedence.
- The precedence of multiplication and division is higher than that of addition and subtraction.
- If an expression contains several operators of the same precedence, then the expression is evaluated from left to right.

In addition to the basic operators, JavaScript also supports modulo (%) operator. This operator performs division and returns the remainder as result. For example, 23%7 is equal to 2.

Unary Operators

While operators like addition and subtraction need to operands to execute, some operators require only one. An example of such operators is *typeof*, which returns the datatype of the data. For example, *typeof 12* returns ‘number’. Please note that ‘+’ and ‘-‘ can also be used as unary operators. This is the case when they are used to specify the sign of a number.

Logical Operators

Three logical operators are available for use in JavaScript namely, Not (!), Or (||) and And (&&). The results of operations that involve these operators are Boolean (true or false). The results of these operations are computed in accordance with the following:

AND (&&)	Binary operator Both the conditions must be true
OR (||)	Binary operator At least one of the conditions must be true
NOT (!)	Unary operator The value of the condition determined is complemented.

For example,

‘Red’ != ‘Blue’ && 5 > 1 = ‘true’

‘Red’ != ‘Blue’ && 5 < 1 = ‘false’

‘Red’ == ‘Blue’ && 5 < 1 = ‘false’

For conditional operators, JavaScript uses short-circuit evaluation. In other words, if the value of the first condition is computed to be ‘false’, the system does not evaluate the other condition and directly presents the result.

Writing Code in JavaScript

Any statement followed by a semicolon is referred to as a statement. This statement may or may not produce a value unlike expressions, which must inadvertently produce a value.

Variables

Manipulation of data is performed with the help of variables. Data is stored in the memory

and a named reference to this memory location is called a variable. Any identifier is declared as a variable by preceding it with a keyword *var*. A sample declaration of a variable is:

var result;

This statement declares a variable *result*. You may also define the variable as you declare it using a statement like this:

var result = 0;

or

*var result = 23*4+6;*

Certain rules have to be followed while naming variables. These rules are as follows:

- A variable name can be a combination of numbers and characters.
- A variable name cannot start with a number.
- The only special characters allowed in variable names is underscore (_) and dollar sign($).
- There should not be any whitespace in the variable name. For example, ‘result value’ is not allowed.
- JavaScript keywords are reserved and cannot be used.

Please note that JavaScript, unlike HTML is case sensitive. Therefore VAL and val are two different variables. Also, it is recommended that the variable name should be such that it describes the purpose for which it is being used. For example, if we name a variable result, it is evident that this variable will contain the result value computed by the code.

Another convention used in JavaScript is to name variables such that the first letter of the variable name is lowercase. However, every subsequent word in variable name starts with a capital letter. An example of this is the variable name, arrayResult. Besides this, the use

of underscore and dollar sign is discouraged. However, they are used in jQuery objects.

Environment

A complete set of variables and the values they contain form what is called the environment. Therefore, whenever you load a new webpage in the browser, you are creating a new environment. If you take the example of Windows 8, it creates an environment when an application starts and the same is destroyed when the application ends.

Functions

A set of statements that solve a purpose are referred to as a function. The purpose of using functions is code reuse. If your program uses functionality multiple times in the program, then it is implemented as a function, which can be called as and when required. Since, a function is to be called from within the code, parameters can be sent to the function from the code. Upon execution, the function returns a value to the calling function. The syntax for function declaration and definition is as follows:

```
function multiply(a, b){

return a*b;

}
```

The name of the function must always be preceded with the keyword function. The variables a and b are parameters passed into the function and the function return the value obtained by computing a*b. This is a simple function, but you can implement complex and large function depending upon the functionality desired.

Now that you have implemented the function, you must be wondering as to how the function is called. Here is an example:

```
var x=2;
```

var y=5

var c=multiply(x, y);

Here, x and y are arguments that the function multiply will receive as parameters.

JavaScript is a loosely typed language. What that means is that if you pass more arguments to a function than what it is expecting, the system simply uses the first n arguments required and discards the rest. The advantage of this functionality is that you can use already implemented functions and pass the extra argument to scale the function and add functionality to it. On the other hand, you will not be able to get any indication of error if you unintentionally pass the wrong number of arguments.

JavaScript also provides some built-in functions for interacting with the user. These functions are as follows:

- alert

  This function raises an alert with a message and the system resumes operation after the user clicks on the OK button. Sample implementation:

  alert(‘Alert message!’);

- prompt

  This function presents a textbox to the user and asks him or her to give input. You can supply the default value in the presented textbox and the user can click on the OK button to accept that the value entered is correct. Sample implementation:

  var result = prompt(‘Enter a value’, ‘default value’);

- confirm

  This message gives the user the choice to OK or CANCEL an action. Sample implementation:

```
var result = confirm(‘Do you wish to proceed?’);
```

Function Scope

Each variable that you declare possesses a scope of operation, which is the function within which the variable has been declared. This is called the local scope. Unlike, many other languages, which define local scope by the curly braces within which the variable lies, JavaScript’s local scope is same as function scope.

In addition to this, JavaScript also supports the concept of global scope, in which variables can be declared global and thus, can be used anywhere in the program.

Nesting Functions

Functions can be nested at any level. In other words, a function can be called from within another function, which could have been called from a different function. However, it is important to note that the scope of variable is within the function in which they are declared.

Conversion of One Data Type to Another

The prompt function discussed in the previous function returns a string. However, you had asked the user to enter a number. In such a scenario, a string to number conversion may be required. In JavaScript, a variable can be converted from one type to another using the following functions:

- Number Function

  This function converts the object supplied to it into number data type. However, if the function is unable to perform the conversion, *NaN* is returned.

- String Function

  This function converts the object supplied to it into string data type.

Conditional Programming

While writing code, you will faced with several situation where you need to execute a different set of instructions if the condition is true and another set of instructions if the same is false.

if-else

In order to implement such scenarios, you can use the if-else construct.

Syntax:

```
If(condition)
{
//code
}
else
{
//code
}
```

Consider a scenario in which you ask the user to enter his or her age using prompt function. Now, you must validate if the age is a valid number, before performing any computation on the value supplied. This is an ideal scenario of implementing conditional programming. Sample implementation for this scenario is:

```
var userAge = prompt('Enter your age: ', '');
if(isNaN(userAge))
{
```

```
alert(‘Age entered is invalid!’);
}
else
{
//code
}
```

In this sample code, the if clause checks if the entered value is a number. If the condition is true, that is the object entered is not a number, the user is given an alert message. However, if the condition is false, the code for else is executed.

It is important to note here that for single statements, it is not necessary to use curly braces. The above mentioned code can also be written as:

```
var userAge = prompt(‘Enter your age: ’, ‘’);
if(isNaN(userAge))
alert(‘Age entered is invalid!’);
else
//code
```

However, it is a good practice to use curly braces as there is scope of adding additional code later on.

if-else if

Another conditional programming construct is if-else if construct. This construct allows you to declare multiple conditions and the actions associated with them. The syntax is:

```
if(condition)
```

```
{
//code
}
else if(condition)
{
//code
}
else
{
//code
}
```

Switch

Multiple else ifs can be implemented using this construct. The execution overhead is high for this conditional programming construct as conditions are sequentially checked for validity. As an alternative, another keyword, switch, is available, which implements multiple conditions in the form of a jump table. Therefore, the execution overhead for switch is lesser than that of if-else if.

Sample implementation:

```
var userChoice = prompt(‘Choose an alphabet: a, b, c’, ‘e’);
switch (userChoice) {
case ‘a’:
alert(‘a chosen\n’);
```

```
break;

case ‘b’:

alert(‘b chosen\n’);

break;

case ‘c’:

alert(‘c chosen\n’);

break;

default:

alert(‘None of the alphabets chosen\n’);

break;

};
```

The switch construct matches that value entered by the user with the values presented in the cases. If a matching value is found, the case is executed. However, in case, none of the case values match the entered input, the default case is executed. Besides this, you can also use conditions in case values and the case for which the condition holds true is executed.

If you do not use the break statement after the code of a case, all the cases following the matching case will be executed. For example, if the user enters ‘b’ for the above example and there are no break statements after the case code, then the output will be:

```
b chosen

c chosen

None of the alphabets chosen
```

Also, it is a good practice to use a break statement in the default case as well.

Note:

If you wish to determine if a keyword has been assigned any value or not, you can use the following code:

```
if(c)
{
//code
}
else
{
//code
}
```

If the variable c has been assigned a not-null value, then the if condition is true and the corresponding code is executed. On the other hand, if the value of variable c is undefined or null, the code within the else construct is executed.

Note:

The value of the following conditions will always be true:

```
” == 0
null == undefined
‘123’ == 123
false == 0;
```

Please note that JavaScript converts the type of the variable concerned for compatibility in

comparisons.

However, if you want to compare both the value and type of two variables, then JavaScript provides another set of operators, === and !===. When the comparisons for the values mentioned in the above example are done using this operator, the resultant will always be false.

Implementing Code Loops

Looping is an important and commonly used construct of any programming language. You will be faced by several situations when you need to perform the same set of instructions, a given number of times. In order to implement this scenario, loops have to be used. Loop constructs available in JavaScript include for, while and do-while.

The while loop includes a condition and as long as the condition remains true, the loop continues to execute. The do – while loop is a modification of the while loop. If the condition in the while is false, the while loop will not execute at all. On the other hand, even if the while condition is false, the do-while loop executes at least once.

Syntax for while loop:

```
while(condition)
{
//code
}
```

Syntax for do-while loop:

```
do
{
//code
```

}

while(condition)

The last type of loop available in JavaScript is for loop. The for loop allows you to initialize the looping variable, check for condition and modify the looping variable in the same statement.

Syntax:

for(initialize; condition; modify)

{

//code

}

Sample code:

for(i=0; i<10; i=i+1)

{

//code

}

This loop will run 10 times.

Note:

1. If at any point in time, you wish the loop to break, you can use the break statement.
2. If you do not specify a condition or specify a condition that is always true, the loop will run infinitely.

Error Handling

Exceptions are expected to occur at several points in your code. Therefore, it is best to

implement a mechanism that can help you deal with these exceptions and avoid crashing.

An exception can be described as an illegal operation or any condition that is unexpected and not ordinary. A few examples of exceptions include unauthorized memory access.

You can perform exception handling at your own level by validating the values of variables before performing any operations. for instance, before performing division, it is advisable to check if the value of the denominator is equal to zero. Any operation that involves division of a number by zero raises the divide-by-zero exception.

However, there are other situations that cannot be handled in this manner. For instance, if the network connection breaks abruptly, you cannot do anything to pre-emptively handle the situation. Therefore, for situations like these, try, catch and finally keywords are used.

The code that is expected to throw an exception is put inside the try block. This exception, upon its occurrence, is caught by the catch block, which executes code that is supposed to be executed immediately after an exception is caught. The catch may also be followed by the finally block, which performs the final cleanup. This block is executed after the execution of try and catch blocks.

Syntax:

```
try
{
//code
}
catch(exception name)
{
//code
```

}

finally

{

//code

}

Working with Objects

JavaScript allows user access to a number of existing objects. One of these objects is an array. This section discusses all the basics related to this chapter. Dealing with objects in JavaScript also includes creation and handling of customized objects. However, this topic shall be covered in the chapter on JavaScript and jQuery.

Arrays

A collection of similar objects that are sequenced contiguously are referred to as an array. This array is given a name and each element can be accessed using the indexer, in the following form:

Let arrName[] be an array of names. The element arrName[2] refers to the third element of the array.

An array can be created using the following three methods:

- Insertion of Items Using Indexer

  An array can be created using the new keyword and then, elements can be added into the array by assigning values to independent elements of the array. The new keyword creates an instance of the object Array using the constructor for the same.

  Sample implementation:

  var arrName = new Array();

arrName [0] = ‘Jack’;

arrName [1] = ‘Alex’;

- Condensed Array

  The second type of implementation also uses the new keyword. However, in this case, the values are assigned to the elements as arguments to the constructor of the Array object.

  Sample implementation:

  var arrName = new Array(‘Jack, ‘Alex’);

- Literal Array

  In this type of array definition, values are provided within the square brackets.

  Sample implementation:

  var arrName = [‘Jack, ‘Alex’];

The advantage of using the first type of definition is that it allows you to assign values to the elements anywhere in the code. On the other hand, the second and third type of implementation requires you to have the exact list of elements with you beforehand.

There are some properties associated with all object. The one property that can come in handy to you is *length,* which is a read-only value and when called return the number of elements present in the array. You can use this property in loops and conditions.

Objects can also have their own functions. These functions are called methods. The methods available for Array include:

- concat

  Returns an array, which is the concatenation of the two arrays supplied to it.

- indexOf

Finds the location of the element concerned in the array and returns the index of the same.

- join

  This method concatenates all the values present in the array. However, all these values are separated by a comma by default. You can specify a delimiter of your choice as well.

- lastIndexOf

  This method works similarly as indexOf. However, it performs the search from the last element of the array. Therefore, it returns the index of the last element that matches the specified criterion.

- pop

  Removes the last element and returns its value.

- push

  Adds the concerned element to the end of the array and returns the changed value of length.

- reverse

  This method reverses the order of the array elements. The original array is modified by this method.

- shift

  Removes and returns the first value. If the array is empty, then undefined is returned.

- slice

  This method requires two arguments, start index and end index. A new array is

created with elements same as the elements present at indexes (start index) and (end index – 1).

- sort

  This method sorts the elements and modifies the original array.

- splice

  This method removes and adds elements to the specified array. The arguments taken by this method are start index (index from where the system should start removing elements), number of elements to be removed and elements to be added. If the value passed for number of elements is 0, then no elements are deleted. On the other hand, if this value is greater than the size of the array, all elements from the start index to the end of the array are deleted.

- toString

  This method creates a string, which is a comma separated concatenated string of all the elements present in the array.

- unshift

  This method adds an element at the first location of the array and return the modified value of length.

- valueOf

  This method returns a string, which is the concatenated, comma-separated string containing all the values present in the array.

Note:

1. When working with functions, you can pass the whole array (using the array name) or a particular element of the array (using array name[indexer]).

2. Array elements can be modified by accessing the element using the indexer. For example, arrName[1] = ‘Danny’; assigns the value ‘Danny’ to the second element of the array.

DOM objects

The primary objects that you need to access while building an application, are the DOM objects associated with it. This access is necessary for you to control and get notified about events that are occurring on the webpage.

The DOM is a representation of a hierarchy of objects. These objects can be accessed using the *document* variable, which is built-in. This variable references the DOM and performs a search, which may return an active or static *NodeList.* While the active NodeList contains a list of elements that keep changing, the static NodeList contains elements that do not change over time. Since the retrieval of the static NodeList takes longer, it is advisable to choose search methods that work with active NodeList.

The search methods available for DOM include:

- getElementById

  This method returns a reference to the first element that has the specified ID.

- getElementsByTagName

  This method returns the active NodeList, which has the specified tag name.

- getElementsByName

  This method returns the active NodeList, which has the specified name. This is a preferred method for option buttons.

- getElementsByClass

  This method returns the active NodeList, which has the specified class name.

However, this method is not supported by Internet Explorer version 8 and earlier.

- querySelector

  This method accepts CSS selector as parameter and return the first matched element. However, this method is not supported by Internet Explorer version 7 and earlier.

- querySelectorAll

  This method accepts CSS selector as parameter and return all the matched elements. Therefore, it returns a static NodeList. However, this method is not supported by Internet Explorer version 7 and earlier.

Events

If you look at JavaScript as an engine, then events are what give it the required spark. Events can occur in two situations. The first type of events are those that occur during user interactions. A user may click an image or enter text. All these are classified as events. Besides this, changes in state of the system are also considered an event. For instance, if a video starts or stops, an event is said to have occurred. The DOM allows you to capture events and execute code for the same.

In JavaScript, events are based on the publish-subscribe methodology. Upon creation of an object, the developer can publish the events that are related to this object. Moreover, event handlers can be added to this object whenever it is used. The event handler function notifies the subscribed events that the event has been triggered. This notification includes information about the event like location of the mouse and key-presses, in addition to several other details relevant to the event.

Capturing events:

There may be situations when an event may be attached to a button click, which may lie

inside a hyperlink. In this situation, there is nesting of elements. Therefore, the event, when triggered, is passed down the DOM hierarchy. This process is called event capturing. However, once the event has reached the element, this event is bubbled up the hierarchy. This process is called event bubbling. This movement of the event across the hierarchy gives the developer an opportunity to subscribe or cancel the propagation, on need basis.

Subscribing to event:

The function, addEventListener, can be used for the subscription process. This function requires three arguments, the event, the function that needs to be called for the event and a Boolean value that determines if the function will be called during the capture or bubble process (true – capture, false – bubble). Mostly, this value is set to false. This is the preferred method for subscription as it is mentioned in the W3C standard.

Sample Code:

```
var btn = document.getElementById(‘btnDownload’);

btn.addEventListener(‘click’, initiateDownload, false);
```

However, other methods also exist, which include giving an online subscription to the html tag. This subscribes the event to the bubble process. The advantage of using this method is that it is the oldest and most accepted method. therefore, you can be sure that this method will work, regardless of what browser you are using. Please see the tag below to understand how this can be done.

```
<button id=‘btnDownload’ onclick=‘initiateDownload();’ >Download</button>
```

You can also use the traditional subscription process that uses JavaScript for subscribing the event.

```
var btn = document.getElementById(‘btnDownload’);
```

```
btn.onclick = initiateDownload;
```

Unsubscribing:

Events can be unsubscribed using the function, removeEventListener, which takes the same set of parameters as addEventListener. For the btn variable used in the previous example, this can be done in the following manner:

```
var btn = document.getElementById('btnDownload');

btn.removeEventListener('click', initiateDownload, false);
```

How to cancel propagation?

The function, stopPropagation, is used for performing this operation. This can be done in the following manner:

```
var btn = document.getElementById('btnDownload');

btn.addEventListener('click', initiateDownload, false);

function initiateDownload (e){

//download

e.stopPropagation();

}
```

How to prevent the default operation?

This can be done by using the function, preventDefault, in the following manner:

```
var hyperlink = document.getElementById('linkSave');

hyperlink.addEventListener('click', saveData, false);

function saveData(e){

//save data
```

```
e.preventDefault();
}
```

JavaScript also provides the *this* keyword, which can be used if you wish to access the event causing element, on a frequent basis.

Window Event Reference

The current browser window is represented by the window variable, which is an instance of the Window object. The following events are associated with this object:

- afterprint
- beforeonload
- beforeprint
- error
- blur
- haschange
- load
- message
- focus
- online
- offline
- pageshow
- pagehide
- redo
- popstate
- storage
- resize
- unload

- undo

Form Event Reference

The actions that occur inside an HTML form trigger the flowing events:

- change
- blur
- focus
- contextmenu
- forminput
- formchange
- invalid
- input
- submit
- select

Keyboard Event Reference

The keyboard triggers the following events:

- keyup
- keypress
- keydown

Mouse Event Reference

The mouse triggers the following events:

- click
- drag
- drop
- scroll

- dblclick
- dragenter
- dragstart
- dragend
- dragover
- dragleave
- mousemove
- mousedown
- mouseover
- mouseout
- mousewheel
- mouseup

Media Event Reference

Media elements like videos, images and audios also trigger events, which are as follows:

- canplay
- abort
- waiting
- durationchange
- canplaythrough
- ended
- emptied
- loadeddata
- error
- loadstart
- loadedmetadata
- play

- pause
- progress
- playing
- readystatechange
- ratechange
- seeking
- seeked
- suspend
- stalled
- volumechange
- timeupdate

Chapter 4: Basics of CSS3

Cascading Style Sheets or CSS provide the presentation that webpages are known for. Although, HTML is capable of providing a basic structure to the webpage, CSS offers developers host of design options. Besides this, it is fast and efficient, which makes it an all more popular design tool.

CSS is known to have evolved from SGML (Standardized Generalized Markup

Language). The goal of efforts made in this direction was to standardize the manner in which web pages looked. The latest version of this technology is CSS3, which is a collection of 50 modules.

The most powerful characteristic of CSS is its cascading ability. Simply, it allows a webpage to take its styles from multiple sheets in such a manner that changes to the style in subsequently read sheets overwrite the style already implemented from one or more of the previous sheets.

How to Define and Apply Style

The definition and application of a style involves two facets or parts, selector and declaration. While the selector determines the area of the webpage that needs to be styled, the declaration block describes the style specifications that have to be implemented. In order to illustrate how it works, let us consider the following example,

```
body {
color: white;
}
```

In this example, the selector selects the body of the webpage and the declaration block defines that the font color should be changed to white. This is a simple example and declarations and selectors can be much more complex than this.

How to Add Comments

Comments can be added to the style sheet using the following format:

```
/*write the comment here*/
```

How to Create an Inline Style

Every element has an associated global attribute, style. This global attribute can be manipulated within the tag for that element to modify the appearance of that element. This type of styling does not require you to specify the selector. Only the declaration block is required. An example of how this is done is given below:

```
<body style='color: white;'>
</body>
```

This HTML tag performs the same functionality as the CSS code specified in the previous section. The advantage of using this approach is that the style information given in this manner overwrites any other styling information. Therefore, if you need to use different

style for one element while the rest of the document needs to follow a different style, then you can use a stylesheet for the document and specify the style for this element in its tag.

How to Use Embedded Style

Another approach for accomplishing the same outcome as inline styles is to use the <style> element within the element concerned, for defining its style specification. Here is how this can be done:

```
<!DOCTYPE html>
<html xmlns='http://www.w3.org/1999/xhtml'>
<head>
<title></title>
<style>
body {
color: white;
}
</style>
</head>
<body>
</body>
</html>
```

How to Create External Style Sheet

For usages where you wish to use the same style for the complete webpage or a number of webpages, the best approach is to use an external style sheet.

This external style sheet can be linked to the HTML page in the following manner:

```
<!DOCTYPE html>

<html xmlns='http://www.w3.org/1999/xhtml'>

<head>

<title></title>

<link rel='stylesheet' type='text/css' href='Content/mainstyle.css' />

</head>

<body>

</body>

</html>
```

You must create a file mainstyle.css, in the Content folder, and put the style rule specified below into the file.

```
body {

color: white;

}
```

Defining Media

It is important to note that a style sheet can contain as many style rules as you want. Besides this, you can also link different CSS files for different media. The different media types are as follows:

- all
- embossed
-

- braille
- print
- handheld
- speech
- screen
- tv
- tty

The media used can be defined in the following manner:

<link rel=‘stylesheet’ type=‘text/css’ href=‘Content/all.css’ media=’all’ />

Defining Character Encoding

You can also define the character encoding used, using the following format:

Style sheet:

Place the following line above the style rule in the style sheet.

@charset ‘UTF-8’;

HTML page:

You must place this line above the link element.

<meta http-equiv=‘Content-Type’ content=‘text/html;charset=UTF-8’ >

Importing style Sheets

As your web pages becomes complex, the style sheets used shall also grow in complexity. Therefore, you may need to use many style sheets. You can import the style rules present in one style sheet to another by using:

@import url(‘/Content/header.css’);

Here, header.css is imported and the url gives the relative address of the style sheet to be

imported.

Importing Fonts

Fonts can be imported using the following format:

@font-face {

font-family: newFont;

src: url(‘New_Font.ttf’),

url(‘New_Font.eot’); /* IE9 */

Selectors, Specificity and Cascading

Selectors can be of three types, class selectors, ID selectors and element selectors. The element selector type is the simplest and requires you to simply name the element that needs to be used. For instance, if you wish to change the background color of the body, then the element selector used is *body.*

While declaring any element, you can assign an ID to it using the id attribute. You can use this ID prefixed with a # as a selector. For example, if you have created a button with ID btnID, then the ID selector for this will be #btnID. Similarly, you can assign a class name to an element using the class attribute. Class name can be used prefixed by a dot(.) in the following manner, .className.

However, if you wish to select all the elements of the webpage, then asterisk (*) to it.

Using Descendent and Child Selectors

You may wish to apply a particular style to a descendant of a selector. This can be done by specifying the complete selector change. It can be done in the following manner:

li a {

text-color: black;

}

On the other hand, you may want to apply to an element only if it is a direct child of the selector. This can be implemented by specifying the parent and child separated by a greater than (>) sign, in the following manner:

li > a {

color: white;

}

Pseudo-element and Pseudo-class Selectors

Now that you know how to apply styles to specific elements, let us move on to implementing styles to more specific sections like the first line of the second paragraph. In order to style elements that cannot be classified on the basis of name, content or is not a part of the DOM tree can be styled using pseudo-classes. The available pseudo-classes include:

- :visited
- :link
- :hover
- :active
- :checked
- :focus
- :nth-last-child(n)
- :not
- :only-child
- :nth-child(formula)
- :lang(language)
- :first-of-type

- :only-of-type

If you want to access information of the DOM tree that is not accessible otherwise, you can use pseudo-elements. Pseudo-elements include:

- ::first-letter
- ::first-line
- ::after
- ::before

Grouping Selectors

Multiple selectors can be used for a style rule. These selectors must be separated by commas. Sample implementation:

```
body, button {
color: white;
}
```

Using Adjacent Selectors

If you want to style the first heading in a div or any similar adjacent elements, the selector is constructed using a plus sign (+) between the two selectors. Sample implementation:

```
div + h1 {
color: white;
}
```

Sibling Selectors

Sibling selectors are similar to adjacent selectors except for the fact that all the matching elements are styled as against adjacent selectors, which only style the first matching element. The selector is constructed using a ~ sign between the two selectors. Sample

implementation:

```
div ~ h1 {

color: white;

}
```

Using Attribute Selector

This selector selects all the elements for which the specified attribute exists. The selector is written in this form:

```
a[title]
```

This selector will select all the links for which the title attribute has been specified. Moreover, this selector type can be modified into attribute-value selector by specifying the attribute value in the following manner:

```
a[title = value]
```

In-Built Styles of Browsers

Every browser has a built-in stylesheet, which is applied to all the webpages opened using this browser. In fact, this stylesheet is applied before any other style sheet. You can define your own style sheet for the browser using the Accessibility option in Tools. However, user style sheets are browser specific. Therefore, if you open a different browser, the style sheet you defined may not be accessible.

In case, you want your user-defined stylesheet to override any other style specified in the HTML page, then you can use the '!important' modifier. This modifier sets highest priority for the specified style statement. Sample implementation:

```
body {

color: white !important;
```

}

Cascading of Styles

The precedence and priority of the styles are decided on the basis of the following parameters.

- Importance
- Specificity
- Textual Order

Working with CSS Properties

Now that you are thorough with the use of selectors, the next step is to look at CSS properties.

Color

One of the most crucial properties that are used in a web page is color, which can be defined using ARGB, RGB and color names.

RGB value are typically defined using a decimal number, which lies between 0-255.

- white #ffffff
- red #ff0000
- black #000000
- green #008000

Color values can also be used instead of the color name. An example of how this can be used is given below.

```
body {

color: #ffffff;

}
```

Another way to specify the color is using the RGB function, which specifies the values of parameters using a number between 0-255 or percentage. Example of this type of declaration is given below:

h1 { color: rgb(255,0,0); }

Other ways to specify color are RGBA, which accepts 4 values and HSL, which defines values for hue, saturation and lightness.

Transparency

The transparency or opacity are defined by a value between 0.0 (invisible) and 1.0 (opaque).

Text

As far as text is concerned, font-face and font-size can be specified. These properties can be defined in the following manner:

h1 { font-family: arial, verdana, sans-serif; }

h1 { font-size: 12px; }

The CSS Box Model

The CSS Box Model assumes that a webpage can be considered to be made up of boxes. The spacing between these boxes are given by margins and padding settings. These properties can be given values in the following manner:

margin: 15px;

padding: 25px;

border: 10px;

Positioning elements

The element used for creating page layouts is . Although, HTML5 recommends the

use of semantic markup instead of div elements, there are still used for content that cannot be styled using semantic markup. A div element can be imagined as a rectangular block and is declared in the following manner:

<!—other elements are enclosed within this area—>

Properties used to define the position of a div element include:

- The position of the div element can be defined using the properties, top, bottom, left and right, in pixels.
- A property, position, is used to define if the position specified is static or relative.
- The float property can be used to allow elements to float to the right or left and is defined as float: left or float: right.
- The clear property places a clear element right after the floating element.
- You can also change the manner in which the browser calculates width with the help of the box-sizing property. This property can take three values: content-box (default setting), border-box and padding-box.

Centering Content

If you are using a fixed width, the div element can be centered using the properties, margin-left and margin-right. If you fix the width and set the margins to *auto*, the extra space on the margins is equally divided. It can be done in the following manner:

#container {

width: 850px;

margin-left: auto;

margin-right: auto;

}

Chapter 5: HTML5 Explained

The chapter focuses on the basics of HTML5 and how they can be used for creating high performance, new-generation pages. However, most of the elements explained in that chapter included elements that HTML5 has received from its predecessors. This chapter takes you a step further in your quest of learning HTML5, introducing concepts that are newly added to this technology.

In the previous chapter on CSS, we introduced the concept of element to you. This element is preferred over all its alternatives as far as page layout creation is concerned. While some people also use the <table> element, it is usually not a recommended solution as it is difficult to maintain as well use. However, both the concepts are elaborated upon this chapter.

Semantic Markup

The and elements are the most commonly used and recommended elements for positioning and formatting content. However, it is recommended that you should use different elements for specifying different sections of the page like header and footer. This shall allow you to position them individually and in a more organized manner. Therefore, the W3C has named these new elements with names like <footer> and <header>.

Browser Support

It is true that your HTML code will not be read by any of your users. However, there are other tools and machines that are constantly parsing your code. These tools include web crawlers, which indexes webpages for searching and NVDA (Non-Visual Desktop Access) devices, which are used by many users with special needs as an alternative for reading and comprehending web pages. Therefore, they require you to use tags that are

understandable.

How to Create HTML5 Documents

Although, the above discussion clearly mentions the importance of using meaning tags and prohibits the use of tags like and , you may still have to use them for styling purposes. As you read on, you will realize how semantic tags must be supplied for providing meaning to your tags. It is important to mention here that semantic tags should be used carefully, and if you realize that there is a need to define custom elements, then go ahead and use the and elements. However, be sure to add an ID and class-name to the elements that describe their meaning as well as possible.

How to Create HTML5 Layout Container

A layout container, as the name suggests, is a container that entails the layout of a page. In other words, the container contains the different sections of the layout or its children in such a manner that they can be positioned in a flexible manner. As a developer, you can easily distinguish between elements on the basis of their class names and IDs. However, this is not true for browsers.

Therefore, there has got to be a way by which you can ask the browser to interpret elements. For instance, you may want to ask the browser to focus on a certain element upon opening. All this and more can be done with the help of layout containers that express elements in such a manner that they are understandable to both the browser and the user.

Some of the commonly used elements for creating a layout container include:

- *<header>*

  It is used to define the header section or the topmost section of the HTML document. This element can also be defined inside the <article> element.

- *<footer>*

  It is used to define the footer section or the bottom-most section of the HTML document. This element can also be defined inside the <article> element.

- *<nav>*

  It is used to define the section that contains all the navigational links.

- *<aside>*

  This element is generally used for sidebars and separates the content in the <aside> element from the content that is outside this element.

- *<section>*

  This element defines a part of the whole section and is named with the elements <h1>, <h2>, <h3>, <h4>, <h5> and <h6>.

- *<article>*

  This element defines a complete unit of content, which you can copy from one location to another. An example of such a unit is a blog post.

Using Roles

The role attribute can be declared inside the and <aside> elements. The role class hierarchy and the usage of roles for providing specific meanings, as far as far as accessibility is concerned, is defined in WAI-ARIA (Web Accessible Initiative - Accessible Rich Internet

Applications).

One of the parent role classes is the landmark role class, which defines the navigational landmarks on the webpage. The child classes of the parent role class include:

- banner

This defines website specific content that is not expected to change from one webpage to another like headers.

- application

  This defines that the specified area is a web application.

- contentinfo

  This defines the information included in the webpage about the webpage like copyright information. This information is mostly a part of the footer content.

- complementary

  This defines a section of the page that is meaningful when detached from the page as well.

- main

  The main web page content is defined using this child role class.

- form

  This defines the area of the webpage that is expected to take webpage inputs.

- search

  This child role class is used to define the location on the webpage that is used for getting the search query from the user and displaying the results of the search.

- navigation

  The area containing navigational links is a part of this child role class.

These roles can be used for providing meaning. However, the new elements included in HTML5 are meaningful themselves. Yet, there are some utilities that are not available in HTML5 and for these, role attribute can be used.

How to Control format using <*div*> element?

As mentioned previously, the element is essentially invisible and does not provide any meaning to the element. However, if you wish to use it for formatting purposes only, then it is perfect for this purpose.

How to Add Thematic Breaks?

The void element <hr/> can be used for adding thematic breaks, which are used for denoting a transition from one set of content to another.

How to Annotate Content?

There are several elements available for annotation. These include <b> and <i>, which you have been using for ages. However, they have new meanings now, in addition to the style that they denote. For instance, the <b> element denotes the style ‘bold’. In addition to this, HTML5 adds the meaning ‘span of text with special focus, but no change in mood, importance level or implication.’

Although, the use of the bold style makes more sense in this context, but you can still use this element for denoting names of products in reviews or keywords. Similarly, the element indicates the relative importance of content and <i> denotes a change in mood or implication of the content concerned. Besides this, the element is used for text that will be alternatively pronounced by the reader.

How to Use <abbr> for Acronyms and Abbreviations?

In the previous versions of HTML, the <acronym> element was used for this purpose. However, this element has become obsolete and the new element used for this purpose is <abbr>. It is an inline element, which is generally used with other inline elements like and .

Element - <address>

This element is used for defining the contact information of the owner or the author of the webpage.

Quotations and Citations

You can indicate that a particular text is a quote by using the element <blackquote>, which is used for a long quotation, and <q>, which is used for an inline quotation. You can mention the source of the quotation using the cite attribute or the <cite> element. However, using the <cite> element inside <q> and <blackquote> elements is considered a better approach. Please remember that the <cite> element can only mention the name of the work and other information elements like author's name and location of publishing, are not included here.

How to Document Code in HTML5?

There are two elements, <code> and <samp>, are used for documenting code. While the element <code> is used for documenting the source code, the element <sample> is used for the output of the code. A sample HTML for how this is done is given below:

<code class="maintainWhiteSpace">

echoContent('Screen');

function echoContent(name)

{

alert('This is' + name + '.');

}

</code>

<samp class="maintainWhiteSpace">

This is Screen.

</samp>

The <pre> Element

It is important to mention here that these elements do not preserve the whitespace. Therefore, a class needs to be implemented for this purpose. This class should look like:

style rule.

.maintainWhiteSpace {

white-space: pre;

}

Some browsers may not support this style rule. Therefore, for such browsers, it is advisable to use the element <pre>. Therefore, <pre> element can be used for defining the pre-formatting rules.

The <var> Element

This element is used to declare that the text specified inside it, is a variable. Example:

The variable <var>i</var> represents the number of iterations for the loop to perform.

The <*br* /> and <*wbr* /> Elements

The
 element implements a line break. On the other hand, the <wbr/> implements a word break.

The <dfn> Element

There may be occasions when you wish to define a term. This can be done using the <dfn> element, which takes title as one of its attributes.

Working with Figures

Images and figures are an integral part of any web page content. Therefore, every figure can also be viewed as a unit of content, which may consist of a caption and a reference from the page to which it may belong. In order to define one or more images, the <figure> element is used. The element <figurecaption> can be used for defining the caption of the figure.

However, it is important to mention here that the <figure> element does not give any importance to the position of the figure and the same is included along with the content. However, if the position and location of the figure is of importance to you., then you must consider using the element.

The <summary> and <details> Elements

The element <summary> contains the summary of the content of the webpage, which can be displayed in the form of a collapsible list using the <details> element. Therefore, when you load a page, only the contents of the <summary> element will be displayed, while the contents of the <details> element are displayed when the user clicks on the summary.

Other Annotations

In addition to the above mentioned, there are a few more annotations available, which include:

- <*s*> - Used for striking out text
- <*u*> - Used for underlining text
- <*mark*> - Used for highlighting text
- <*ins*> - Used for indicating that the text has been inserted
- <*del*> - Used for indicating that the text has been deleted
- <*small*> - used for indicating that the text must be printed in fine letters
- <*sub*> - Indicates that the text is a subscript

- <*sup*> - Indicates that the text is a superscript
- <*time*> - Used for indicating that the text denotes time and date
- <*kbd*> - used for indicating that the text is a user input

Language Elements

You may need to use characters of Chinese, Japanese or Korean origin in your text. In order to support this inclusion, the element <ruby> can be used. Inside this element, other elements like <bdi> and <bdo>, for defining the isolation and direction of text. Besides this, <rt> and <rp> elements can also be used for placing notation or parentheses in the text of <ruby> element.

Working with Lists

In HTML5, several elements for defining unordered, ordered and descriptive lists exist. A fourth category of 'Custom lists' is also present to allow customization by the developer. The list items for all these are declared using the <li> element. Moreover, all the three types of lists support list nesting.

Ordered Lists

Ordered lists are declared using the element <ol> and the elements of 'order' in this list are brought about by an automatic numbering of the elements that are included in this list. The attributes that can be used with ordered lists include:

- start - Used to set the starting number of the list
- reversed - Used for declaring if the list has to be ordered in an ascending or descending order
- type – Used for declaring the type of the list, which can be A, a, 1 or I.

Unordered Lists

This type of a list is declared using the <ul> element and there is no numbering of

elements in this case. The elements of the lists are simply represented as bullet points.

Description Lists

This type of a list is declared using the <dl> element. Using this element, you can give a description containing zero or more terms. Besides this, the elements of the list are declared using the <dt> element, which specifies the term, and <dd>, which gives a description of the term.

Custom Lists

The developer can make use of CSS styles to create custom lists. In this case, a different style rule can be created for each level of a nested list.

Working with Tables

Another format for arranging and presenting data in webpages is tables. Tables are declared using the <table> element and represents data in a rows-columns format. The cells of the tables are defined using the <tr> and <td> elements. While <tr> is used for rows, <td> is used for columns.

Despite that fact that HTML5 tables are one of the most powerful constructs available to the developer, it is important to understand how and where tables can be most appropriately used. Here are the reasons why tables should not be used:

- The table created for a web page is not rendered until the </table> tag is read. On the other hand, if the same construct is created using the element, the content will be rendered as it is read.
- Tables are extremely difficult to maintain.
- Tables are difficult to interpret for accessibility devices.

Sample Implementation:

<table>

```
<tr>
<td>Frank</td>
<td>1978</td>
</tr>
<tr>
<td>David</td>
<td>1967</td>
</tr>
<tr>
<td>Alex</td>
<td>1989</td></tr>
</table>
```

The table created above will look like:

Frank	1978
David	1967
Alex	1989

As you observe the table, you must have realized that the table is not complete unless we define what is called the table headers. This can be done using the <th> element. You can create table headers both vertically and horizontally. For example, we can define the following table header in the code used above.

```
<table>
<tr>
```

```
<th>Name</td>

<th>Year of Birth</td>

</tr>

<tr>

<td>Franky</td>

<td>1978</td>

</tr>

<tr>

<td>David</td>

<td>1967</td>

</tr>

<tr>

<td>Alex</td>

<td>1989</td></tr>

</table>
```

The resulting table for this code will look like this:

Name	Year of Birth
Franky	1978
David	1967
Alex	1989

In the above case, the table headers are simply of a larger font size. However, you can

style these as you want by style rules. This can be done in the following manner:

```
th {
background-color: black;
color: white
}
```

This style rule will color the cells of the table headers with black color and the text will be written in white.

The normal behavior of most browsers is to automatically place all the <tr> elements in the <tbody>, indicating that this is the body of the table. However, it is a good practice to define the <tbody> explicitly. Besides this, the <thead> and <tfoot> can also be explicitly defined. The header, body and footer of the table can be individually styled using CSS. As a rule, you can have one header element, one or more body elements and one footer element.

The next important content feature that must be added to the table created above is the table caption. In order to define the table caption, the <caption> element is used.

In some cases, you may feel the need to style individual columns. This may seem like a difficult task considering the fact that tables are essentially row centric in HTML. While you have <tr> elements to identify rows, there are no <tc> elements for identifying columns. The <td> element identifies a cell. However, you can still style individual columns using the <colgroup> or <col> elements. The <table> element can have the <colgroup> element, which includes the <col> elements for columns that are a part of this group of elements. In addition, the <col> element also has a *span* attribute for defining the columns that are a part of this group. A sample implementation to explain how this works is given below:

<colgroup>

<col span=“2” class=“vHeader” />

</colgroup>

The CSS style rule for styling this group of columns can be something like this –

.vHeader {

color: red;

}

While the tables discussed till now are regular tables, HTML5 also supports irregular tables, which can be described as tables that have a different number of columns for each row. The rowspan and colspan attributes can be used for managing the layout of the table.

Chapter 6: JavaScript and jQuery

You must have got a hang of the power of JavaScript already. In the chapter on JavaScript, you have already learnt how to create and adding JavaScript for making web pages dynamic. The biggest challenge of web development is to design webpage elements that can run just as well on any browser as different browsers provide support for different elements, making it difficult to find a common ground.

This chapter takes you a step further in JavaScript development by teaching you how to create objects and use them. Besides this, it also introduces you to the concept of jQuery, which attempts at creating browser-compatible code. Although, it cannot promise 100% browser compatibility, but it certainly solves the day-to-day issues regarding the same.

How to Create JavaScript Objects

Anything from numbers to strings are objects in JavaScript. Therefore, it is essential to

know how to create and deal with these effectively. The simplest way to create objects in JavaScript is using the object literal syntax. The following example illustrates how it is done.

```
var customer1 = {

yearOfBirth: 2000,

name: ‘Alex’,

getCustomerInfo: function () {

return ‘Customer: ‘ + this.name + ‘ ‘ + this.yearOfBirth;

}

};
```

This code creates an object customer1, with the data members, name and yearOfBirth and the member function getCustomerInformation. It is also important to note the use of *this* keyword, which accesses the values correctly being used or referenced for the object concerned.

Besides this, you can also create objects dynamically using the new keyword. The methods inherited include:

- constructor
- isPrototypeOf
- hasOwnProperty
- toLocalString
- propertyIsEnumerable
- valueOf
- toString

Once the object has been created, properties can be added to the same in the following

manner:

```
function getCustomer(myName, myYearOfBirth) {

var newCust = new Object();

newCust.name = myName;

newCust.yearOfBirth = myYearOfBirth;

newCust.getCustomerInfo = function () {

return ‘Customer: ‘ + this.name + ‘ ‘ + this.yearOfBirth;

};

return newCust;

}
```

This code creates an object newCust dynamically. Several instances of this can be created in the following manner:

```
var cust1 = getCustomer (‘Alex’, 1978);

var cust2 = getCustomer (‘David’, 1986);
```

Although, JavaScript doesn’t support a particular keyword ‘class’, but you can simulate classes using the method mentioned above.

Namespaces

There is no specific keyword like *namespace* for implementing namespaces. However, namespaces can be implemented using the concepts of classes and objects. If you classify variables and methods into objects and access them as instances of these objects, then you are placing only the names of these objects in the global namespace, reducing the scope of the variables to the object that they belong.

Implementing Inheritance

You can define ‘is-a’ relationships between objects in JavaScript by creating objects and then classifying those objects on the basis of their common characteristics. For instance, if you are implementing an object for employee of a company. You can create objects for specific types of objects like managerTechnical, managerGeneral, technicalStaff, recruitmentStaff and officeStaff and then classify them into objects, technicalStaff, which includes the objects managerTechnical and technicalStaff, and adminStaff, which includes the managerGeneral, recruitmentStaff and officeStaff. In a similar manner, new functions can also be defined.

Working with jQuery

JQuery is a library of browser-compatible helper functions, which you can use in your code to minimize the efforts required for typing, implementation and testing. These functions are essentially written in JavaScript. Therefore, you can also call jQuery, a JavaScript library.

The list of functionalities that are available in jQuery include:

- Attributes, which are a group of methods that can be used for getting and setting attributes of the DOM elements.
- Ajax, which is a group of methods that provide support for synchronous and asynchronous server calls.
- Core Methods are the fundamental jQuery functions.
- Callbacks object is an object provided for management of callbacks.
- Data Methods are methods that facilitate the creation of association between DOM elements and arbitrary data.
- CSS Methods are methods that can be used for getting and setting CSS-related properties.

- Dimensions are methods that can be used for accessing and manipulating the dimensions of DOM elements.
- Deferred object is an object that is capable of registering multiple callbacks while maintaining the data of state change and propagating the same from one callback to the next.
- Forms are methods that are used for controlling form-related controls.
- Traversing, this is a group of methods that provide support for traversing the DOM.
- Effects are methods that can be used for creating animations for your webpage. Events are methods used to perform event-based execution.
- Selectors are methods that can be used for accessing elements of DOM in CSS selectors.
- Offset are methods that are used to position the DOM elements.
- Utilities, which is a group of utility methods

Before getting to use jQuery, you will need to include it into your project. Once you have installed it and you are ready to use it to your project, the next step is to learn how to use it.

First things first, you need to reference the jQuery library on the webpage that needs to use it in the following manner:

<script type="text/javascript" src="Scripts/qunit.js"></script>

<script src="Scripts/jquery-1.8.2.js"></script>

The next thing to know is that the jQuery code that you are hoping to use in your HTML page lies in the jQuery namespace, which has an alias $. Therefore, you can write either jQuery.jFeature or $.jFeature when referring to a feature of jQuery.

Before, you can start using it in your webpages, you will also need to change the default.js file as follows:

```
function initialize() {

txtInput = $(‘#txtInput’);

txtResult = $(‘#txtResult’);

clear();

}
```

This allows you to use jQuery and CSS selectors by matching them using their IDs.

Also, as you move ahead with coding using jQuery, remember to refresh the screen using Ctrl+F5 after making any changes as the browser may not be able to catch the JavaScript modification right away. Moreover, use jQuery objects as much as possible because the cross-browser compatibility that they offer.

A DOM object can be referenced from a jQuery wrapper in the following manner:

```
var domElement = $(‘#txtInput’)[0];
```

Here is a simple code that checks if the element exists before referencing it.

```
var domElement;

if($(‘#txtInput’).length > 0){

domElement = $(‘#txtInput’)[0];

}
```

How to Create a jQuery wrapper for Referencing a DOM element

A jQuery wrapper can be created from a DOM element reference in the following manner:

```
var myDoc = $(document);

var inText = $(this).text();
```

The first statement assigns the wrapped object to the variable. On the other hand, the

second statement wraps the object referenced using *this*.

How to Add Event Listeners

jQuery provides the .on method for subscribing to events. Besides this, you can unsubscribe using the .off method. These methods can be used in the following manner:

$(‘#btnSubmitInfo’).on(‘click’, onSubmit);

$(‘#btnSubmitInfo’).off(‘click’, onSubmit);

How to Trigger Event Handlers

JQuery provides triggers or the method, triggerHandler, for triggering event handlers or handler code execution. This can be done in the following manner:

$(‘#btnSubmitInfo’).triggerHandler(‘click’);

Initialization Code

You will often be faced with the requirement to run an initialization code upon the loading of an HTML document. You can do this using jQuery in the following manner:

<script>

$(document).ready(function () {

initializationFunction();

});

</script>

This can be placed at the bottom of the HTML document. It will call the initializationFunction.

CHAPTER 7: FORMS

In the previous chapters, you have already studied how HTML documents can be created

and manipulated. Taking a lead from them, we can now move on to understanding forms, which is one of the most crucial and commonly used units of content in webpage development.

Simply, a form is a way in which data is collected and sent to a location where it can be processed, which is a server in most cases. However, since we are yet to discuss server side scripting, we will focus on sending the data to an email address. However, it is important to note that we do not recommend this practice and it is used only for understanding purposes.

Web Communications

Before moving to the working of forms, it is important to know some basics of HTTP. A typical web communication consists of the following activities:

1. When a user browses a webpage, a request for access to a web server resource is initiated by sending a GET HTTP request.
2. The request is processed by the server and sends a response to the browser using HTTP protocol.
3. The browser process the server response and presents it to the user in the form of a ‘form’.
4. The user enters inputs to the form and upon hitting the submit or enter button, this data is sent to the server, using HTTP protocol again.
5. This data is again processed by the server and the server posts its response to the browser, which is displayed on the webpage.

Web Servers

Originally, web servers were designed to receive and process requests and send the results back to the browser using HTTP. Initially, when the web pages were simple, such web servers were able to process a good number of requests per unit time. There were no states

involved as sending and receiving requests were as simple as opening a connection, transferring data and closing the connection.

The new age web servers are much more equipped that these simple web servers. The web servers of today implement what is called the ‘keep alive’ features, which ensures that the connection remains open for a definite period of time in which subsequent requests by the same browser to the server can be entertained.

Web Browsers

The web browser is a desktop application, which displays web pages and manages user interactions between the webpages and the server. The communication between the web servers and pages is established using technologies like AJAX and Asynchronous JavaScript.

How is Data Submitted to the Web Server

An HTML form can be created using the <form> element in the following manner:

<form method=“post” action=“getCustomerInformation.aspx” >

Enter Customer Number:

<input type=“text” name=“Number” />

<input type=“submit” value=“Get Customer Information” />

</form>

This form takes in the customer number and returns a page that displays the details of the customer.

However, it is important to note that not all elements can be used for submitting data in a form. The allowed elements for this purpose are:

- <textarea> - It takes a multi-line input

- <button> - It is a clickable button, which can be placed on any content or image.
- <select> - It is a drop-down list, which allows multiple selections. The selected options can be identified using jQuery: $(‘option:selected’)
- <input type=’checkbox’> - It is a checkbox, which has a value attribute used for setting as well as reading the status of the checkbox. The jQuery for identifying checked checkboxes is: $(‘input[type=checkbox]:checked’)
- <input type=’button’> - It is a clickable button, which has a text prompt.
- <input type=’datetime’> - It is a control, which is date and time (UTC).
- <input type=’date’> - It is a control, which is date-only.
- <input type=’email’> - It is a file-select field, which has a browse button for uploading a file.
- <input type=’color’> - It is a color picker.
- <input type=’hidden’> - It is a hidden input field.
- <input type=’datetime-local’> - It is a control, which is date and time (any timezone).
- <input type=’month’> - It is a month-year control.
- <input type=’image’> - It is an image submit button.
- <input type=’password’> - It is a password field, with masked characters.
- <input type=’number’> - It is a numeric field.
- <input type=’range’> - It is a control, which accepts a numeric value and defines the allowed range of values.
- <input type=’radio’> - It is an option button, which has a value attribute for setting and reading the status of the button. The jQuery used for identifying the marked radio buttons is $(‘input[type=radio]:checked’)
- <input type=’search’> - It is a text field, which is used for entering the search string.

- <input type=’reset’> - It is a button that can be used for resetting the fields of a form.
- <input type=’url’> - It is a URL field.
- <input type=’tel’> - It is a telephone number field.
- <input type=’submit’> - It is a submit button.
- <input type=’time’> - It is a control, which accepts a time value.
- <input type=’text’> - It is a single-line text field.
- <input type=’week’> - It is a week-year control.

The <*label*> Element

It is the element that is used to convey the meaning of an element to the user. The text in this element is displayed inside the textbox, which is auto-removed when the user clicks on it. You can also specify the style of a label.

Specifying Parent Forms

There may be situations where the submission elements of a form may not lie inside the same construct. Therefore, gathering data, in this case, can be quite a challenge. In order to address this issue, HTML5 provides an attribute, id, which can be set for multiple form elements. This shall allow data collection from different form elements in one go.

How to Trigger Form Submission

Upon triggering, all the data collected from the submission elements of the form or forms of the same id is sent to the server using an HTTP method. The <input> element can be used for triggering form submission. Besides this, you can also use JavaScript for this purpose. In order to implement this, you must give an id to the concerned form, myFirstForm. The default.js file, which is linked to the HTML document, must contain the following code:

```
$(document).ready(function () {
```

```
$(‘#myFirstButton’).on(‘click’, submitMyFirstForm);

});

function submitMyFirstForm() {

$(‘#myFirstForm’).submit();

}
```

If the method attribute of the form is not given any value, then it is set to a default GET. Moreover, the action attribute will also have a default value. Therefore, the button click will reference the page to same page. However, the URL will now include a QueryString, which is a combination of values selection or entered by the user. For instance, if the form requests the user to enter Customer ID and the user enters 1245, then the QueryString will be:

customerID=1245

This QueryString will be appended to the URL in the following manner:

Mywebpage.asp? customerID=1245

It is also important to mention here that the QueryString is URI encoded. Therefore, special characters are represented by specific values. For instance, a space is represented as ‘+’ and exclamation mark (!) as ‘%21’. Name-value pair is represented as ‘name=value’ and name-value pairs are separated by ‘&’.

How to Serialize the Form

You can serialize the form using the jQuery serialize method. This can be done in the following manner:

```
var myFormData = $(‘#myFirstForm’).serialize();
```

You can decode the URI-encoded string using the following:

var data = decodeURIComponent(myFormData);

Using Autofocus Attribute

By default, the focus is not set to any element of the form. However, you can set focus using the focus method, which can be implemented in the following manner:

$(‘input[name=“firstName”]’).focus();

However, you can also set autofocus in the following manner:

<input type=“text” name=“firstName” autofocus=“autofocus”/>

How to Use Data Submission Constraints

A form can send data only if it satisfies the following constraints:

- Name attribute must be set.
- You must have set the value of form submission element.
- The <form> element must have its form submission element defined and form submission elements should not be disabled.
- If multiple submit buttons have in implemented, the values will be submitted only on the click of the activated submit button.
- Select the check boxes
- Select the Option buttons
- The <option> elements must have set <option> elements.
- If there is a file selection field, one of the fields must be selected.
- The declare attribute of the object elements must be set

Always remember that the reset buttons don’t send any data and the form need not have an ID for its data to be sent.

How to Use POST or GET

There are two HTTP methods available for submitting data to the server. These methods

are GET and POST. In the former case, the URL is appended with the QueryString. However, in case of the latter, the information is sent within the message body.

Form Validation

It is beneficial to understand that the root of all security issues in web application is user data. The moment you decide to open your application to user data and interactions, you are making your application vulnerable to security threats. Therefore, you need to validate any data that you receive before processing it to prevent any issues from cropping up. Validation can be provided at the browser or server end. However, server side validation is recommended as browser-level validation can be easily manipulated.

The simplest form of validation that you can implement is using the required attribute in the <select> element. You can set this attribute in the following manner:

<select name=“dateOfBirth” required=“required”>

The validation error generated is browser dependent and each browser has a different method of communication to the user that a field is required for submission.

Besides this, the placeholder attribute is also available, which keep the prompt fixed on the unfilled field until a value for the same is provided by the user. It can be implemented in the following manner:

<input type=“text” name=“Date of birth” required=“required” placeholder=“enter the date of birth”/>

The addition of time, date and type based inputs in HTML5 makes validation much simpler as they can directly be matched to see if they have valid input values or not. Besides this, email, numbers and ranges can also be easily validated.

HTML5 performs validation and matches it with the :valid or :invalid pseudoclasses. If the validation is successful, the value is matched to :valid, else it is matched to :invalid.

However, if a value is not ‘required’, it is matched to :optional pseudoclass.

Chapter 8: Web Services

All the chapters discussed till now dealt with browser level coding and scripting. However, now it is time to move on to server side scripting. This chapter focuses on the use of JavaScript at the server level, which is possible with the help of Node.js, and how you can work around with web services.

Basics of Node.js

Node.js is a platform, which is made on Google Chrome, and can be used for creating scalable and flexible applications. It allows you to write JavaScript code for the server. However, before you can begin, you must download and install Node.js on your system.

Writing a Basic Code

The first step is to open any text editor and create a file named myFile.js. In the file, write the code:

```
var http = require(‘http’);

http.createServer(function (request, response) {

response.writeHead(200, {‘Content-Type’: ‘text/plain’});

response.end(‘Hello World!\n’);

console.log(‘Handled request’);

}).listen(8080, ‘localhost’);

console.log(‘Server running at http://localhost:8080/’);
```

The first line loads the http module while the second line creates a server object. The function createServer takes in two parameters, request and response. All the website handling is done from these functions. In this example, the response function ends by

writing ‘Hello World’ on the screen.

The function createServer, returns a server object, which call the function, listen. This function listens at the port 8080 and the IP address of the host is set to 127.0.0.1. therefore, if there is a network adapter installed on your system, your web server will start listening to web requests rights away. The last line prints a line on the screen to let the user know that the server is running and listening to requests.

Once you have created the file and saved the contents of the file as mentioned above, you must open the command prompt and write the command:

Node myFile.js

Now, keeping the command prompt active, you must open the web browser and type the address: http://localhost:8080/

As soon as the request is sent and a response is received, the same is communicated to the user using the console window. If you have been able to do this successfully, then you have just created your first node.js website. If you wish to stop the running of the code, you can just press Ctrl+C.

Now that you know how requests are received, it is time to look at how these requests are processed and responses are generated. You may need to use the *url* module for parsing the QueryString.

The code mentioned below shows how you can parse the URL string and generate a response in accordance with it.

```
var http = require(‘http’);

var url = require(‘url’);

http.createServer(function (request, response) {

var url_parts = url.parse(request.url, true);
```

```
response.writeHead(200, {‘Content-Type’: ‘text/plain’});

response.end(‘Hey ‘ + url_parts.query.name + ‘.\n’);

console.log(‘Handled request from ‘ + url_parts.query.name);

}).listen(8080, ‘localhost’);

console.log(‘Server is running at: http://localhost:8080/’);
```

You can test the running of this code in the similar manner as the previous code.

How to Create Node.js Module

You can create modules by writing code in the form of functions and then, calling these modules from the main code.

```
var myHttp = require(‘http’);

var myUrl = require(‘url’);

function start(){

http.createServer(function (request, response) {

var url_parts = url.parse(request.url, true);

response.writeHead(200, {‘Content-Type’: ‘text/plain’});

response.end(‘Hello ‘ + url_parts.query.name + ‘!\n’);

console.log(‘Handled request from ‘ + url_parts.query.name);

}).listen(8080, ‘localhost’);

console.log(‘Server running at http://localhost:8080/’);

}

exports.start = start;
```

when you save this piece of code in a file, a module is created. This module can be used by other functions using the require function. For instance, if the file is saved as sample1.js, then the start() can be used in another function using:

var samplex = require('./sample1.js');

sample1.start();

How to Create a Node.js package

A collection of modules is referred to as an application. Once you have published your package, it can be installed and used. Consider for example, a package of different mathematical modules.

The root folder must have the following:

README.md

\packName

\lib

main.js

\bin

mod1.js

mod2.js

Creating Aggregate Module

You may wish to make only one object for the user to access. The user should be able to access all the modules of the package through this object. In order to accomplish this, a main.js module must be created in the bin folder that must define the modules to be included in the module.exports.

How to Create README.md File

The README.md file is a help file that can be used by the developer as a startup guide for using your package. The extension of this file is .md, which is a short form for mark-down. This format gives readability to the text written in this file.

A sample file of this type is given below:

samplePackage package

==========

In samplePackage, the following functions are available:

- **add** Performs addition of two numbers and presents the result.

- **sub** Performs subtraction of one number from the other and presents the result.

How to Create package.json File

This file contains the metadata for the package and can be created manually using the command:

npm init

This command creates the file, which can later be edited. A sample file is given below:

{

“name”: “sampleFile”,

“version”: “0.0.0”,

“description”: “This is a sample file “,

“main”: “bin/main.js”,

“scripts”: {

“test”: “echo \“This is a test file\” && exit 1”

},

“repository”: ””,

“keywords”: [

“sample”,

“example”,

“add”,

“sub”

],

“author”: “XYZ”,

“license”: “ABC”

}

In addition to test scripts, you can also give git scripts, which are the best available source control managers.

How to Publish a Package

As mentioned previously, a package can be defined in terms of a folder structure. When you publish your package, you make it accessible to all users. In order to perform this operation, you must use the npm command, which is also the command used for searching and installing packages. However, you shall be required to create an account for yourself before publishing any of your packages using the command: npm adduser. After you enter the required information, your account is created. However, what this also means us that there is no validation required. Therefore, anyone can add code to the repository. Therefore, you should be careful while downloading and installing packages from the registry.

In order to publish a package, you simply need to go to the root directory of the package

and enter the command npm publish in the command prompt.

How to Install and Use the Package

A package that is published can be downloaded and installed by any user. You simply need to go to the folder and give the command, npm install samplePackage. This installs the package locally. On the other hand, if you wish to install the package globally, you can give the command, npm install –g samplePackage. For a global installation, you will need to create a link from each application to the global install using the command, npm link samplePackage.

The route to a global install is a junction. You can get into the node_modules folder and back using the cd command. Once you are inside the folder, you can give the command: npm install contoso, to initiate an install. You can now write some code that uses the package. A sample is given below:

```
var samplePackage = require('samplePackage');

var finalResult = 0;

console.log();

finalResult = samplePackage.add (5,10);

console.log('add (5,10) = ' + finalResult);

console.log();

result = samplePackage.sub (50,10);

console.log('sub(50,10) = ' + finalResult);

console.log();

console.log('done');
```

This code tests the modules of the package. You can execute the code using:

node main

The package can be uninstalled locally using:

npm uninstall samplePackage

However, if you wish to uninstall the package globally, you can do it using

npm uninstall -g samplePackage

How to Use Express

1. The first step is to install Node.js and create a sample for keeping all .js files and projects. Besides this, you must also install Express, which is a web application framework for Node.js development.
2. You can create a package using the following set of commands and instructions.

1. npm init
2. You can create myPackage.js file containing the following contents:

```
{
“name”: “Sample”,
“version”: “0.0.0”,
“description”: “This is a sample website.”,
“main”: “main.js”,
“scripts”: {
“test”: “echo \“Error: Test not specified\” && exit 1”
},
“repository”: ””,
“author”: “XYZ”,
```

“license”: “BSD”

}

“private”: true,

“dependencies”: {

“express”: “3.0.0”

}

}

In order to use the file in Express, dependencies have to be added. Moreover, if you do not define it to be private, you may get an error from the firewall of your computer as it tries to load the page.

3. Give the install command: npm install

4. You can use the command, npm ls, to see if the package is present in the registry.

4. You can create a simple application using the following set of instructions:

0. Create a file myApp.js and add the following to the file:

var express = require(‘express’);

var app = express();

2. You can define the route using the myApp.Method() syntax.

app.get(‘/’, function(request, response){

response.send(‘Hey World!’);

});

The code mentioned above will send the response ‘Hey World!’ as and when a request is received.

3. The last section of code that must be added is for listening to the request. This code is as follows:

   var port = 8080;

   app.listen(port);

   console.log('Listening on port: ' + port);

4. Once the file is complete, you can save it and run it using the command, node app. So, now if you open the browser and enter the address http://localhost:8080/, you will get the response *Hey World!*.

5. You can add webpages to applications by replacing the app.get statement with app.use(express.static(__dirname + '/public')); This will allow you to use the same code for a number of webpages. Sample implementation of this concept is given below:

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title></title>

</head>

<body>

<form method="get" action="/submitHey">

Enter First Name: <input type="text" name="firstName" />

<input type="submit" value="Submit" />

</form>

</body>
```

```
</html>
```

Please note that the action attribute is set to /submitHey. In other words, this resource is called at the server for handling the data that is passed to it using the QueryString. The myApp.js file should contain the following:

```
var express = require('express');

var app = express();

app.use(express.static(__dirname + '/public'));

app.get('/SubmitHey', function (request, response) {

response.writeHead(200, { 'Content-Type': 'text/html' });

response.write('Hey ' + request.query.userName + '!<br />');

response.end('Enjoy.');

console.log('Handled request from ' + request.query.userName);

});

var port = 8080;

app.listen(port);

console.log('Listening on port: ' + port);
```

The app can be run in the manner mentioned above.

5. The *formidable* package can be used for posting back data. While the previous method used the GET method, this method uses the POST method.

1. To illustrate how it works, create an HTML as follows:

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
<title></title>
</head>
<body>
<form method=”post” action=”/SubmitHeyPost”>
Enter Name: <input type=“text” name=“firstName” />
<input type=“submit” value=“Submit” />
</form>
</body>
</html>
```

2. Now, in the command prompt, you need to a give a command for retrieving the formidable package, which is:

   ```
   npm info formidable
   ```

3. You can also modify the package.jason file in the following manner:

   ```
   {
   “name”: “HelloExpress”,
   “version”: “0.0.0”,
   “description”: “Sample Website”,
   “main”: “index.js”,
   “scripts”: {
   “test”: “echo \“Error: test not specified\” && exit 1”
   ```

```
},
“repository”: ””,
“author”: “XYZ”,
“license”: “BSD”,
“private”: true,
“dependencies”: {
“formidable”: “1.x”,
“express”: “3.0.0”
}
}
```

4. Now you can install the *formidable* package by typing the following command into the command prompt:

```
npm install
```

   This command installs the package locally. Therefore, you will need to add a line to myApp.js that allows the file to reference the package:

```
var formidable = require(‘formidable’);
```

5. A sample myApp.js file shall look like this:

```
var express = require(‘express’);
var app = express();
var formidable = require(‘formidable’);
app.use(‘/forms’, express.static(__dirname + ‘/public’));
app.post(‘/SubmitHeyPost’, function (request, response) {
```

```
if (request.method.toLowerCase() == ‘post’) {
var form = new formidable.IncomingForm();
form.parse(request, function (err, fields) {
response.writeHead(200, { ‘Content-Type’: ‘text/html’ });
response.write(‘Hey ‘ + fields.userName + ‘!<br />’);
response.end(‘Enjoy this POST.’);
console.log(‘Handled request from ‘ + fields.userName);
});
}
});
app.get(‘/SubmitHey’, function (request, response) {
response.writeHead(200, { ‘Content-Type’: ‘text/html’ });
response.write(‘Hey ‘ + request.query.userName + ‘!<br />’);
response.end(‘Enjoy. ‘);
console.log(‘Handled request from ‘ + request.query.userName);
});
var port = 8080;
app.listen(port);
console.log(‘Listening on: ‘ + port + ‘port’);
```

6. You can now run the application in a similar manner as you did for the previous example.

Working with web services

One of the biggest drawbacks of a typical website scenario is that the HTML page is repainted even if the new page is the same as the previous page. This causes you to lose bandwidth and resources. This drawback can be addressed using web services, which can be used for sending and receiving data, with the benefit that the HTML page is not repainted. The technology used for sending requests is AJAX or Asynchronous JavaScript and XML. This technology allow you to perform the data sending operation asynchronously.

Before moving any further, it is important to know the basics of web services and how they can be used. A client needs to communicate with the web server on a regular basis and this communication is facilitated by the web service. In this case, the client can be anyone from a machine using the web service to the web service itself. Therefore, the client, regardless what it is, needs to create and send a request to the web service, and receive and parse the responses.

You must have heard of the term mashups, which is a term used to describe applications that pierce together web services. Two main classes of web services exist, which are arbitrary web services and REST or representational state transfer. While the set of operations are arbitrary in the first case, there exists a uniform operations set in the second.

Representational State Transfer (REST)

This framework uses the standard HTTP operations, which are mapped to its create, delete, update and retrieve operations. Moreover, REST does not focus on interacting with messages. Instead, its interactions are focused towards stateless resources. This is perhaps the reason why REST concept is known for creation of clean URLs. Examples of REST URLs include http://localhost:8080/Customers/2, which deletes a customer and

http://localhost:8080/Vehicles?VIN=XYZ12, which is used to retrieve the information about a vehicle for which a parameter is passed using GET method.

Some firewalls may not allow the use of POST and GET methods. Therefore, it is advisable to use ‘verb’ in the QueryString. An example of how the URL will look like is:

http://localhost:8080/Vehicles?verb=DELETE&VIN=XYZ987

The HTTP protocol also allows you to implement security using the HTTPS version. REST provides several benefits like easy connection, faster operation and lesser consumption of resources. However, many developers prefer to use JSON or (JavaScript Object Notation) because it is compact in size. Besides this, REST only supports GET and POST, which restricts its capabilities, Therefore, some developers switch to RESTFUL.

Arbitrary Web Services

This type of web services is also referred to as big web services. An example of such services is WCF or Windows Communication Foundation. Arbitrary web services expand their realm of operations by not mapping their operations to only aspects of the protocol. As a result, they provide more functionality, which include many security mechanisms and message routing.

This type of web services possess a typical interface format, which can be used by the client for reading and parsing information. As a result, the client can make calls to the service immediately. A common API format is the Web Services Description Language (WSDL). In case of arbitrary web services, the client must assemble its request with the help of a SOAP (Simple Object Access Protocol) message. This web service does not use the HTTP protocol and instead uses the TCP.

How to Create RESTful Web Service using Node.js

In the example mentioned previously, the samplePackage can be exposed as web service.

The GET method can be used on the package and the operation is passed as a parameter. A good RESTful implementation of this package can look something like this:

http://localhost:8080/samplePackage?operation=add&x=1&y=5

How to Use AJAX to Call Web Service

Web services can be called asynchronously using AJAX, which is in actuality a JavaScript. Instead of making a call to the server and repainting the HTML document, AJAX just calls back to the server. In this case, the screen is not repainted. A sample implementation is given below:

A MyPage.html can be created with the following code:

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<title></title>

<script type="text/javascript" src="/scripts/jquery-1.8.2.min.js"></script>

<script type="text/javascript" src="/scripts/default.js"></script>

</head>

<body>

<form id="myForm">

Enter Value of X:<input type="text" id="x" />

Enter Value of Y:<input type="text" id="y" />

Result of Operation:

<button id="btnAdd" type="button">Add the Numbers</button>

</form>

</body>

</html>

The default.js file must be modified to contain the code required for processing these functions. Be sure to check the version of jQuery and whether it matches the version name that you have mentioned in your default.js file. The <form> element used here is only a means of arranging the format of data and the data is not actually sent via the form to the server. The JavaScript and jQuery access the data entered and perform the AJAX call.

How to Use XMLHttpRequest

The object that actually makes an AJAX call is XMLHttpRequest, which can be used for sending/receiving XML and other types of data. This object can be used in the following manner:

var xmlhttp=new XMLHttpRequest();

xmlhttp.open(“GET”,”/add?x=50&y=1”,false);

xmlhttp.send();

var xmlDoc=xmlhttp.responseXML;

The first line creates the object while the second line sets up the use of GET method with the specified QueryString and the use of ‘false’ indicates that the operation must be performed asynchronously. The next line sends the request and the last line sets the response to a variable, which can be later read and parsed for processing the response.

However, the output generated is JSON and not XML, therefore, the default.js file must be changed to:

$(document).ready(function () {

```
$(‘#btnAdd’).on(‘click’, addTwoNum)

});

function addTwoNum() {

var x = document.getElementById(‘x’).value;

var y = document.getElementById(‘y’).value;

var result = document.getElementById(‘finalResult’);

var xmlhttp = new XMLHttpRequest();

xmlhttp.open(“GET”, “/add?x=” + x + “&y=” + y , false);

xmlhttp.send();

var jsonObject = JSON.parse(xmlhttp.response);

result.innerHTML = jsonObject.result;

}
```

The code extracts the x and y values from the <input> element for the same. After this, the XMLHttpObject is created and the open method is called using the QueryString. After the execution of the send function, the response string is parsed. In order to test the page, you can give the command:

```
node app
```

This command starts the web service, after which you can open the browser window with the link:

```
http://localhost:8080/SamplePage.html
```

this code is operational now. However, you may wish to perform the AJAX call in an asynchronous manner. For this, you must locate the open method and change the ‘false’

parameter to ‘true’. Besides this, you will also be required to subscribe to *onreadystateschange* for managing asynchronous call. This can be implemented in the following manner:

```
function addTwoNum () {

var a = document.getElementById(‘a’).value;

var b = document.getElementById(‘b’).value;

var result = document.getElementById(‘finalResult’);

var xmlhttp = new XMLHttpRequest();

xmlhttp.onreadystatechange = function () {

if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {

var jsonObject = JSON.parse(xmlhttp.response);

result.innerHTML = jsonObject.result;

}

}

xmlhttp.open(“GET”, “/add?a=” + a + “&b=” + b , true);

xmlhttp.send();

}
```

The codes for states are as follows:

- 0 - Uninitialized
- 1 - Loading
- 2 - Loaded
- 3 - Interactive

- 4 - Completed

If progress events are provided by the server, you can subscribe to the browser’s progress event. Then, an event listener can be added to initiate the execution of the code when the event is triggered. This can be done in the following manner:

```
function addTwoNum () {

var a = document.getElementById(‘a’).value;

var b = document.getElementById(‘b’).value;

var finalResult = document.getElementById(‘finalResult’);

var xmlhttp = new XMLHttpRequest();

xmlhttp.onreadystatechange = function () {

if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {

var jsonObject = JSON.parse(xmlhttp.response);

result.innerHTML = jsonObject.result;

}

}

xmlhttp.addEventListener(“progress”, updateProgress, false);

xmlhttp.open(“GET”, “/add?a=” + a + “&b=” + b , true);

xmlhttp.send();

}

function updateProgress(evt) {

if (evt.lengthComputable) {

var percentComplete = evt.loaded / evt.total;
```

```
//display the progress by outputting percentComplete

} else {

// You need to know the total size to compute the progress

}

}
```

You can also perform error handling by subscribing to the error event and the abort event. This can be done in the following manner:

```
function addTwoNum () {

var a = document.getElementById(‘a’).value;

var b = document.getElementById(‘b’).value;

var finalResult = document.getElementById(‘finalResult’);

var xmlhttp = new XMLHttpRequest();

xmlhttp.onreadystatechange = function () {

if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {

var jsonObject = JSON.parse(xmlhttp.response);

result.innerHTML = jsonObject.result;

}

}

xmlhttp.addEventListener(“progress”, updateProgress, false);

xmlhttp.addEventListener(“error”, failed, false);

xmlhttp.addEventListener(“abort”, canceled, false);
```

```
xmlhttp.open(“GET”, “/addition?x=” + x + “&y=” + y , true);

xmlhttp.send();

}

function transferFailed(evt) {

alert(“An error occurred”);

}

function canceled(evt) {

alert(“canceled by the user”);

}
```

It is also beneficial for you to know that different browsers implement and respond to objects in different manners. Therefore, it is advised that you must XMLHttpRequest as jQuery wrappers. The use of jQuery makes the code browser independent. The jQuery wrapper for AJAX call is $ajax(), which accepts an object as a parameter. The above code can be re-implemented in the following manner:

```
function addTwoNum () {

var a = $(‘#a’).val();

var b = $(‘#b’).val();

var data = { “a”: a, “b”: b };

$.ajax({

url: ‘/add’,

data: data,

type: ‘GET’,
```

```
cache: false,
dataType: ‘json’,
success: function (data) {
$(‘#result’).html(data.result);
}
});
}
```

The variable values are retrieved and supplied to the ajax call. The other properties and objects are set as shown. Another topic of concern is CORS, or Cross-Origin Resource Sharing, which is a way for allowing cross-site AJAX calls.

Chapter 9: WebSocket Communications

There are scenarios that may require the establishment of two-way communication between the client and the server. A sample application that requires this form of communication is a chat application. This chapter gives an elaborative study of how web socket communication can be established and used.

Communicating by using WebSocket

Websocket communications are established using the WebSocket protocol, which uses the TCP connection for establishing a full-duplex communication pathway. The protocol has been standardized by the IETF (RFC 6455) and the working draft of the WebSocket API is in W3C.

This technology has replaced the concept of long polling. In this case, the client sends a request and keeps waiting for a response until a response is received. The benefit of this approach is that the connection remains open and as and when the data is available, it is

immediately sent. However, the connection may be timed out and a re-establishment may be required.

Understanding WebSocket Protocol

The WebSocket protocol allows bi-directional connection establishment and is designed for implementation for web servers and clients. This technology only interacts with the handshake and is interpreted as a transition from the http to WebSocket. The frame of this protocol has no headers and an overhead of 2 bytes. It is a lightweight technology and its high performance allows communication of live content and games.

Defining WebSocket API

The heart of the WebSocket technology is WebSocket object. This object is defined on the window object and contains the following members:

- close
- WebSocket constructor
- binaryType
- send
- extensions
- bufferedAmount
- onerror
- url
- onclose
- onopen
- onmessage
- readyState
- protocol

How to Implement WebSocket Object

The WebSocket communication is based on TCP and thus, it operates on port number 80. You can use any HTML form for demonstration. The WebSocket url is of the ws:// and wss:// form. In order to implement this form, you need to create an object of the WebSocket API, which can then be configured using onopen, onclose, onerror and onmessage events. The default.js file must be modified in the following manner:

```
/// <reference path="_references.js" />

var wsUri = 'ws://echo.websocket.org/';

var webSocket;

$(document).ready(function () {

if (checkSupported()) {

connect();

$('#btnSend').click(doSend);

}

});

function writeOutput(message) {

var output = $("#divOutput");

output.html(output.html() + '<br />' + message);

}

function checkSupported() {

if (window.WebSocket) {

writeOutput('WebSocket is supported!');

return true;
```

```
}
else {
writeOutput(‘WebSockets is not supported’);
$(‘#btnSend’).attr(‘disabled’, ‘disabled’);
return false;
}
}
function connect() {
webSocket = new WebSocket(wsUri);
webSocket.onopen = function (evt) { onOpen(evt) };
webSocket.onclose = function (evt) { onClose(evt) };
webSocket.onmessage = function (evt) { onMessage(evt) };
webSocket.onerror = function (evt) { onError(evt) };
}
function doSend() {
if (webSocket.readyState != webSocket.OPEN)
{
writeOutput(“NOT OPEN: ” + $(‘#txtMessage’).val());
return;
}
writeOutput(“SENT: ” + $(‘#txtMessage’).val());
```

```
webSocket.send($(‘#txtMessage’).val());

}

function onOpen(evt) {

writeOutput(“CONNECTED”);

}

function onClose(evt) {

writeOutput(“DISCONNECTED”);

}

function onMessage(evt) {

writeOutput(‘RESPONSE: ‘ + evt.data);

}

function onError(evt) {

writeOutput(‘ERROR: ‘ + evt.data);

}
```

When you create a WebSocket object, a connection to URI is automatically initiated. The function, connect, subscribes the object the events mentioned above. The message is sent using the doSend function. However, the readyState property of the object is checked before sending any message. The values of the property can be as follows:

- Closed = 3
- Closing = 2
- Open = 1
- Connecting = 0

Upon reception of message from the server, the client calls the onMessage function, which passes the event object. The property, data, of this object, contains the message sent from the server. On the contrary, if an error has occurred, then the onError function is called and the data property contains the reason for error generation. In some cases, the data property may be ‘undefined’. This is particularly the case in scenarios where the connection has timed-out.

How to Deal with Timeouts

A timeout is identified when no activity is triggered for as long as 20 minutes. One of the most effective ways to deal with timeouts is sending an empty message to the server on a periodic basis. This can be done in the following manner.

```
/// <reference path=“_references.js” />

var wsUri = ‘ws://echo.websocket.org/’;

var webSocket;

var timerId = 0;

$(document).ready(function () {

if (checkSupported()) {

connect();

$(‘#btnSend’).click(doSend);

}

});

function writeOutput(message) {

var output = $(“#divOutput”);

output.html(output.html() + ‘<br />’ + message);
```

```
}
function checkSupported() {
if (window.WebSocket) {
writeOutput('WebSockets supported!');
return true;
}
else {
writeOutput('WebSockets NOT supported');
$('#btnSend').attr('disabled', 'disabled');
return false;
}
}
function connect() {
webSocket = new WebSocket(wsUri);
webSocket.onopen = function (evt) { onOpen(evt) };
webSocket.onclose = function (evt) { onClose(evt) };
webSocket.onmessage = function (evt) { onMessage(evt) };
webSocket.onerror = function (evt) { onError(evt) };
}
function keepAlive() {
var timeout = 15000;
```

```
if (webSocket.readyState == webSocket.OPEN) {
webSocket.send(‘’);
}
timerId = setTimeout(keepAlive, timeout);
}
function cancelKeepAlive() {
if (timerId) {
cancelTimeout(timerId);
}
}
function doSend() {
if (webSocket.readyState != webSocket.OPEN)
{
writeOutput(“NOT OPEN: ” + $(‘#txtMessage’).val());
return;
}
writeOutput(“SENT: ” + $(‘#txtMessage’).val());
webSocket.send($(‘#txtMessage’).val());
}
function onOpen(evt) {
writeOutput(“CONNECTED”);
```

```
keepAlive();
}
function onClose(evt) {
cancelKeepAlive();
writeOutput("DISCONNECTED");
}
function onMessage(evt) {
writeOutput('RESPONSE: ' + evt.data);
}
function onError(evt) {
writeOutput('ERROR: ' + evt.data);
}
```

How to Handle Disconnects

Connections may sometime close errors concerning the network. Therefore, you may be required to call the connect function while you are inside the construct of the onClose function. This may lead to issues. A common problem faced in this regard is that the server is unable to identify that this new connection request belongs to a client that already exists. You can handle this issue by sending an identification message to the server.

How to Deal with Web Farms

You may need to run your application in web farm, in which case the incoming requests may be handled by many servers. Multiple server are used to perform load balancing and performance optimization. In this case, you can implement sticky servers, which ensures that a client is served by only one server. In other words, all the requests by one client are

processed by the same server. This helps you to address issues that may arise due to open connections. These problems can also be handled using products like Microsoft App Fabric Caching Service and Redis.

How to Use WebSocket Libraries

Dealing with all the issues mentioned above can be challenging. Therefore, the best way to deal with the situation is to use server and client libraries. There are several libraries like SignalR and Socket.IO are available for you to explore.

Chapter 10: Managing Local Data With the Help of Web Storage

Most of the topics discussed in the previous section focused on improving the performance of the websites and the services that it provides its customers. Therefore, handling information or data has been restricted to the data that is transferred between the client and server. In such a case, the server needs to wait for the complete round trip to occur. This involves cost overheads.

In order to address this issue, most modern day browsers have started supporting web or DOM storage, which allows storage of some data at the client level. This chapter provides an overview of storage mechanisms used and how they can be used to improve the performance of the system in this regard.

Introduction to Web Storage

In most web application, a server side storage solution like SQL Database Server is implemented. This may be an attempt to kill a mosquito with a sword. You don’t need such a huge storage solution as storing small bits of information at the browser level could have solved your purpose.

Http cookies has been the most preferred and commonly used form of data storage at the browser level. In fact, it is used by most browsers for storing user information. The cookie

value can be set in the following manner:

```
function setCookieValue(cName, cValue, daysToExpire) {

var daysToExpire = new Date();

daysToExpire.setDate(daysToExpire.getDate() + daysToExpire);

cValue = cValue + “; expires=” + daysToExpire.toUTCString();

document.cookie = cName + “=” + cValue;

}
```

The cookie value can be retrieved using the following code:

```
function getCookieValue(cName)

{

var cookie = document.cookie.split(“;”);

for (var i = 0; i < cookie.length; i++) {

var cookies = cookie[i];

var indexVal = cookies.indexOf(“=”);

var keyVal = cookies.substr(0, index);

var actualVal = cookies.substr(index + 1);

if (keyVal == cName)

return actualVal;

}

}
```

The last segment of code, shown below, illustrates how cookies can be used.

setCookie(‘fName’, Clark, 1);

var fName = getCookie(‘fName’);

You can also use the cookie plug-in available in jQuery, which can be used in the following manner:

$.cookie(‘fName’, ‘Clark’);

var fName = $.cookie(‘fName’);

Although, cookies are extensively used, they pose some serious issues like overhead and capacity limitation. Some alternatives available include:

- Google Gears
- User Data
- Java Applets
- Flash Player

These options are better cookies and addresses its limitations, but they suffer from some issues, which include:

- Pug-in is required
- User can block them
- Not useful for corporate users
- They are vendor-specific.

HTML5 Storage

Although, these tools offers some good, yet potentially improvable solutions, HTML5 has come up with some innovative tools in their attempt to provide a comprehensive solution.

- **Web storage**

The simplest form of storing data over the web is web storage, in which key-value pairs are stored.

- **Web SQL database**

  If the application requires a full relational database, then Web SQL database is also available for us.

- **IndexedDB**

  This is a NoSQL database that is seen as a good alternative to its relational counterpart. It is considered to be a good option for complex database requirements.

- **Filesystem API**

  This is the preferred form of data storage is the data is of larger data types like images, audios and videos.

In all of the above mentioned types of data storage, the data is tied to the URL. Therefore, this data cannot be accessed by other websites. This is an important consideration when multiple websites are hosted on a shared domain.

Exploring localStorage

LocalStorage is one of the two storage mechanisms used for data storage. It is basically a Storage Object, which is stored as a global variable. The biggest advantage of using this mechanism is that the API provided for reading and writing data is simple and easy to use.

How to Use localStorage Object Reference

The attributes and methods available include:

- **setItem(key, value)**

  It is a method that is used to set a value with the help of an associated key. The

syntax for this operation is:

localStorage.setItem(‘fName’, $(‘#fName’).val());

- **getItem(key)**

  It is a method that is used to get a value associated to the specified key. The syntax for this operation is:

  var fName = localStorage.getItem(‘fName’);

  If no value is set, then null is returned.

- **removeItem(key)**

  It is a method that is used to remove a value associated to the specified key. The syntax for this operation is:

  localStorage.removeItem(‘fName’);

- **clear()**

  Removes all items from the local storage. Its syntax is:

  localStorage.clear();

- **length**

  It is the property that returns the number of entries in the storage.

  var count = localStorage.length;

- **key(index)**

  This method is used for finding the key associated with the given value. The syntax for this operations is:

  var keyValue = localStorage.key(1);

One of the biggest benefits of using this storage mechanism is that it is supported by most

modern browsers, unlike other storage mechanisms, which are supported by only a few browsers. Although, this mechanism is supported by most web browsers, it is a good idea to check it before using it. If the first reference to this storage returns null, then it can be assumed that the storage mechanism is not supported. In addition to this, you can also use the construct given below for this purpose:

```
function isWSSupported() {

return ‘localStorage’ in window;

}

if (isWSSupported()) {

localStorage.setItem(‘fName’, $(‘#fName’).val());

}
```

In addition to this, you can also use the Modernizr javascript library for the checking purpose. It is worth mentioning here that this type of web storage allows 5MB of web storage, as against the 4KB of web storage allowed by cookies. When the storage space is exceeded, an exception for the same is thrown by the system.

Although, web storage allows storage of strings only, but you can also store complex objects with the help of JSON utility methods.

How to Use Short-term Persistence with *sessionStorage*

SessionStorage is also a Storage Object, with the same attributes and methods as localStorage. The only difference between the two storage mechanisms is that sessionStorage stores data for the session only. If the browser window is closed, the data will no longer be available for use.

Issues Related to sessionStorage and localStorage

Some of the issues related to these two storage mechanisms include:

1. Simultaneous reading and writing of the hard drive.
2. Slow searching capabilities.
3. No support for transactions.

How to Handle Storage Events

Synchronization is the biggest challenge faced in the implementation of web storage. In order to deal with this issue, you can subscribe to storage events, which can notify you about any changes that may have been occurred. When you subscribe to a StorageEvent, you get access to the following properties:

- newValue
- oldValue
- url
- key
- storageArea

Unlike other events, cancelling or bubbling a StorageEvent is not allowed. You can subscribe to an event using the following construct:

```
function respond (exc) {

alert(exc.newValue);

}

window.addEventListener('storage', respond, false);
```

This event can be triggered using the following operation:

```
localStorage.setItem('name', Clark);
```

You can also bind to these events with the help of jQuery.

Chapter 11: Offline Web Applications

The previous chapter discussed storage, which is one of the most used offline application. However, in some scenarios, you may require better tools and advanced features like indexing, asynchronous support and handling transactions. These features are a part of the many other offline applications that are available.

This chapter discusses Websql and IndexedDB, which are relational constructs for data storage. However, they cannot be used for storing files. Therefore, FileSystem API is used for this purpose. The chapter ends with a discussion on how you can make your website offline-friendly using HTTP Cache.

Web SQL

Web SQL is a relational database and its most recent implementations are created on SQLite. However, it is important to mention that this database constrict is no longer supported by W3C yet it is still supported by many browsers.

How to Create and Open Database

A database can be created and opened using the following syntax:

var dbase = openDatabase(‘Lib’, ‘1.0’, ‘Sample library’, 5 * 1024 * 1024);

The parameters are as follows:

- name
- version
- displayName
- estimatedSize
- creationCallback

How to Manipulate Database

Manipulation scripts in SQL can be given as parameter to the transaction.executeSql()

function for manipulating the database. An example of how a table can be created is given below:

```
transaction.executeSql(“CREATE TABLE IF NOT EXISTS employees(” +
“id INTEGER PRIMARY KEY AUTOINCREMENT, ” +
“fName TEXT, “+
“lName TEXT, ” +
)”);
```

IndexedDB

Till now, you have dealt with the two extreme cases of storing data over the web. While the web storage mechanisms follows the simple key-value pair model, Web SQL is based on the relational database model. IndexedDB is an alternative model that lies somewhere in between these two models. It is capable of dealing with everything from strings to complex objects. However, while using this storage mechanism, use browser-independent methods.

A database can be opened or created in the following manner:

```
var indexedDB = window.indexedDB;
var openRequest = indexedDB.open(‘Lib’, 1);
```

An important facet of IndexedDB object is the keypath property, which is used to define the attributes that must be used as the key. If the specified property does not exist, then an auto-incrementing or auto-decrementing attribute can be created.

Indexes can be added using the following code:

```
var openRequest = indexedDB.open(‘Lib’, 2);
openRequest.onupgradeneeded = function(response) {
```

```
var store = response.currentTarget.transaction.objectStore(“employees”);

store.createIndex(‘lName’, ‘lName’, { unique: false });

};
```

The parameters required are: Name, keypath and optional parameters. In a similar manner, indexes can be removed using the following code:

```
store.deleteIndex(‘lastName’);
```

You can add and remove object stores storing the following code:

```
var openRequest = indexedDB.open(‘Lib’, 3);

openRequest.onupgradeneeded = function(response) {

response.currentTarget.result.deleteObjectStore(“employees”);

};
```

Similarly, an object can be added in the following manner:

```
response.currentTarget.result.createObjectStore(“employees”, { keypath: ‘email’ });
```

The next set of operations include transactions, which are performed in the following manner:

1. A transaction has to be opened in the following manner:

   ```
   var newTrans = dbase.transaction([‘employees’, ‘payroll’]);
   ```

2. You can add a new record in the following manner:

   ```
   var openRequest = indexedDB.open(‘Lib’, 1);

   var dbase;

   openRequest.onsuccess = function(response) {

   dbase = openRequest.result;
   ```

```
addEmployee();
};
function addEmployee() {
var myTrans = dbase.transaction('employees', 'readwrite');
var employees = myTrans.objectStore("employees");
var request = employees.add({fName: 'Sammy', lName: 'Carnie'});
request.onsuccess = function(response) {
alert('ID of New Record = ' + request.result);
};
request.onerror = function(response);
}
```

3. A record can be updated in the following manner:

```
var openRequest = indexedDB.open('Lib', 1);
var dbase;
openRequest.onsuccess = function(response) {
dbase = openRequest.result;
updateEmployee();
};
function updateEmployee() {
var myTrans = dbase.transaction('employees', 'readwrite');
var employees = myTrans.objectStore("employees");
```

```
var request = employeess.put({fName: ‘Sammy’, lName: ‘Carnie’}, 1);
request.onsuccess = function(response) {
alert(‘ID of Updated Record = ‘ + request.result);
};
request.onerror = function(response);
}
```

4. You can delete a record using the following code:

```
function deleteEmployee() {
var myTrans = dbase.transaction(‘authors’, ‘readwrite’);
var employees = myTrans.objectStore(“employees”);
var request = employees.delete(1);
request.onsuccess = function(response);
request.onerror = function(response);
}
```

How to Work with FileSystem API

The above mentioned storage mechanisms can be used for storing data, which is in form of strings. However, complex data structures like files and images may also be stored using URIs. However, it is an excessively costly option. HTML5 offers a low cost and simpler option called FileSystem API for this purpose. Using this format, you can create files and directories on a user file system’s sandboxed location.

The functionalities available include:

1. A FileSystem can be opened and created using:.

```
window.requestFileSystem(TEMPORARY, 5 * 1024 * 1024, getFile, handleError);

function getFile(fileSystem) {

fileSystem.root.getFile(“sample.txt”, { create: true }, fileOpened, handleError);

}

function fileOpened(fileEntry) {

alert(“File opened!”);

}

function handleError(error) {

alert(error.code);

}
```

The parameters for the requestFileSystem() function are type, size, successCallback and errorCallback. On the other hand, the parameter for getFile() function are path and options.

2. You can write to a file using the following code:

```
function writeToFile(fileWriter) {

fileWriter.onwriteend = function() { alert(‘Success’); };

fileWriter.onerror = function() { alert(‘Failed’); };

fileWriter.write(new Blob([‘Hello world’], {type: ‘text/plain’}));

}
```

You can append to a file using the following code:

```
function writeToFile(fileWriter) {

fileWriter.onwriteend = function() { alert(‘Success’); };
```

```
fileWriter.onerror = function() { alert('Failed'); };
fileWriter.seek(fileWriter.length);
fileWriter.write(new Blob(['Hello world'], {type: 'text/plain'}));
}
```

3. You can read from a file using the following code:

```
function readFile(file) {
var fileReader = new FileReader();
fileReader.onloadend = function() { alert(this.result); };
fileReader.onerror = function() { alert('Failed'); };
fileReader.readAsText(file);
}
```

4. You can delete a file using the following code:

```
fileEntry.remove(fileRemoved, handleError);
```

Similar functions exist for directories as well.

APPENDIX

<a> Hyperlink

<abbr> Abbreviation

<address> Contact information

<area> Image map region

<article> Independent section

<aside> Auxiliary section

<**audio**> Audio stream

<**b**> Bold text

<**base**> Document base URI

<**bb**> Browser button

<**bdo**> Bi-directional text override

<**blockquote**> Long quotation

<**body**> Main content

<**br**> Line break

<**button**> Push button control

<**canvas**> Bitmap canvas

<**caption**> Table caption

<**cite**> Citation

<**code**> Code fragment

<**col**> Table column

<**colgroup**> Table column group

<**command**> Command that a user can invoke

<**datagrid**> Interactive tree, list, or tabular data

<**datalist**> Predefined control values

<**dd**> Definition description

<**del**> Deletion

<**details**> Additional information

<dfn> Defining instance of a term

<dialog> Conversation

** ** Generic division

<dl> Description list

<dt> Description term

**** Stress emphasis

<embed> Embedded application

<fieldset> Form control group

<figure> A figure with a caption

<footer> Section footer

<form> Form

<h1> Heading level 1

<h2> Heading level 2

<h3> Heading level 3

<h4> Heading level 4

<h5> Heading level 5

<h6> Heading level 6

<head> Document head

<header> Section header

<hr> Separator

<html> Document root

<i> Italic text

<iframe> Inline frame

<img> Image

<input> Form control

<ins> Insertion

<kbd> User input

<label> Form control label

<legend> Explanatory title or caption

<li> List item

<link> Link to resources

<map> Client-side image map

<mark> Marked or highlighted text

<menu> Command menu

<meta> Metadata

<meter> Scalar measurement

<nav> Navigation

<noscript> Alternative content for no script support

<object> Generic embedded resource

<ol> Ordered list

<optgroup> Option group

<option> Selection choice

<**output**> Output control

<**p**> Paragraph

<**param**> Plug-in parameter

<**pre**> Preformatted text

<**progress**> Progress of a task

<**q**> Inline quotation

<**rp**> Ruby parenthesis

<**rt**> Ruby text

<**ruby**> Ruby annotation

<**samp**> Sample output

<**script**> Linked or embedded script

<**section**> Document section

<**select**> Selection control

<**small**> Small print

<**source**> Media resource

<**span**> Generic inline container

<**strong**> Strong importance

<**style**> Embedded style sheet

<**sub**> Subscript

<**sup**> Superscript

<**table**> Table

<tbody> Table body

<td> Table cell

<textarea> Multiline text control

<tfoot> Table footer

<th> Table header cell

<thead> Table head

<time> Date and/or time

<title> Document title

<tr> Table row

<ul> Unordered list

<var> Variable

<video> Video or movie

<wbr> Optionally break up a large word at this element

C++ Programming For Beginners

A Simple Start To C++ Programming Written By A Software Engineer

Scott Sanderson

Table of Contents

Appendix

BONUS BOOK OFFER 1

BONUS BOOK OFFER 2

BONUS BOOK OFFER 3

C++ Basics

C++ is an object-oriented language, which derives most of its features and base constructs from C. This is perhaps the reason why C++ is known as a ‘superset’ of C. Any program written in the C programming language is legally and syntactically an ANSI C++ program. However, the reverse of this statement is not true. Without dwelling into a lot of theory, let us directly move on to programming, which is certainly the best way to learn any programming language.

A Basic C+

+ Program

The simplest program to start with is one that prints ‘Hello World’ on the screen. The code for this program is given below:

Code:

```
//Simple C++ Program
#include <iostream>
using namespace std;
int main ()
{
    cout << “Hello World!”;
    return 0;
```

```
}
```

Output:

```
Hello World!
```

Before moving any further, let us analyze this program line-by-line. The first line is the comments line. This line is not included in the actual code, which is run by the machine. It is only for developers to reference at a later time and make understanding of the code easier. There are two ways of writing comments in C++. One of these methods uses '//' before the text and the other one uses '/*' before the text and '*/' at the terminating end.

The second line uses a #include and includes iostream library to the code. This library facilitates input-output operations and allows the user to use console for giving input and getting output. The third statement specifies that the code is using the namespace std. The std namespace contains all the inbuilt libraries of the C++ package. Therefore, if you are hoping to use any these libraries, then it is essential for you to include this statement.

The program for printing 'Hello World!' begins with the fourth statement. The function, main(), is the function from where execution begins in C++ programming language. It is mandatory for the main() to return an integer value. The scope of the function is described using curly braces, '{' and '}'.

The body of the function contains two statements. The first statement uses a standard I/O operation cout, which is used for printing the specified output on the screen. A similar function, cin, is used for taking input from the console. While, the operation, cout, uses '<<' operator, cin operation uses '>>' operator. The next statement returns an integer value, as previously mentioned.

Both the statements are terminated with a semicolon (;), which is a C++ standard. However, this rule does not apply to directives like *include*. It is important to note here that C++ is not format specific. Therefore, you can write all the statements one-after-

another, separated by semicolons, in the same line. Giving a proper structure to the program makes it more readable and easy to understand.

Please note that the instructions for writing, editing, compiling and running a program are compiler-dependent. You may use Dev-C++ or Turbo C if you are using Windows and any GCC compiler for Unix/Linux or Mac.

Data Types, Variables and Constants

While the program illustrated in the previous chapter is rather simple and with questionable usefulness, it should have laid the required foundation to get started with C++ programming. Quite obviously, programming is not just limited printing texts on the console. In fact, it is much more complicated than that. As we dwell into the complexities of this programming language, the first thing to look at is variables.

As a kid, when you learned addition and subtraction, your teacher or parents must have asked you to keep one number in your memory, and then memorize the second number. Once you have the two numbers, you must add them and memorize the result. When we mention the operation ‘memorize’, we are actually referring to a variable in the context of the computer.

Therefore, a variable can simply be described as a memory cell that saves a defined value. For example, if you go back to the example of addition, then you can assume two cells, ‘*a*’ and ‘*b*’, which contain the values to be added. A result called ‘*result*’ shall contain the added value. Therefore, a, b and result are variables.

Identifiers

It is difficult to memorize memory cell addresses and refer to them every time you wish to perform an operation on the same. In order to solve this issue, the concept of identifiers was introduced. An identifier is a name given to the memory cell with the defined value, while it is declared. Any changes, made to the value of this memory cell, are performed using the associated identifier. In the previous example, the names, ‘a’, ‘b’ and ‘result’ are identifiers.

The identifiers used as names for variables should also be valid. There are several

guidelines that determine if an identifier is valid or not, which are as follows:

1. Should be a sequence of letter, numbers and underscores (_).
2. Should not contain spaces, symbols or punctuation marks.
3. Should begin with an underscore or letter.
4. Should not start with a number.
5. Should not be a reserved keyword. A list of reserved keywords is provided at the end of the chapter in Appendix I.

Please note that some compilers may not support keywords starting with underscore. They are reserved for usage by the compiler itself. Also, C++ is case-sensitive. Therefore, x and X are two different variables. So, be careful!

Fundamental Data Types

Going back to the example of addition, you inherently know that you are memorizing two integers. This is not the case with the computer as you need to explicitly declare everything before any processing can take place. Therefore, you need to tell the computer which type of data you are going to store in the variable concerned. It is only after this declaration that the computer assigns memory to the variable.

The table given below shows the fundamental data types, their memory requirements and a brief description of what they entail. However, please note that the memory size is machine and system dependent. The values given in the table are true for a standard 32-bit machine.

Data Type	Memory Required	Descriptions
char	1 byte	Any character
short int or	2 bytes	Short Integer

short		
int	4 bytes	Standard integer
long int or long	4 bytes	Long integer
float	4 bytes	Floating point number
double	8 bytes	Double precision floating point number
long double	8 bytes	Long double precision floating point number
bool	1 byte	Boolean
wchar_t	2 bytes/4 bytes	Wide character

Declaration of Variables

A variable can be declared using the following statement:

<data type> <identifier>

Example:

int a;

This statement declares that the variable a will contain integer values. There can be several variations of this statement. You can define the variable in the following manner using the same statement.

int a = 0;

This declaration assigns the value 0 to the variable a.

In addition, you can also declare several variables using the same statement. An example is given below:

int a, b, c;

In this case, all the variables, a, b and c, are declared to be integers.

Integers (int, short and long) can be declared as signed or unsigned. You need to use the keywords *signed* and *unsigned* in the statement in the following manner:

signed int a;

or

unsigned int a;

In the absence of the keyword, the system assumed that the variable is signed in nature. You can also use the keywords *signed* and *unsigned* with char. If you use *unsigned char*, then you can only store characters in the variable. However, if you use signed char, you can store integers, one byte long, in the variable.

Scope of Variables

As a rule, you can use a variable only after you have declared it. In addition to this, there exists a concept of ‘scope’. Scope is of two types: global and local. A variable is said to have global scope if the variable is declared outside the body of the code. As a result, this variable is available to all the functions of the program.

Any variable declared inside the body of the code is inevitably local in nature and has local scope. The scope limitation of the variable is the block in which the variable is present, which is defined by enclosed braces, ‘{‘ and ‘}’.

Initialization of Variables

When you declare a variable, it contains garbage value. In other words, the value of the variable cannot be predicted. In order to make the variable usable and useful, you need to define it by assigning a value to the same. Initialization of a variable can be done via two methods.

According to the first method, the variable must be defined at the time of its declaration.

Standard format:

<type identifier> <variable> = <value>

Sample implementation:

int a = 0;

The other type of initialization makes use of the constructor method.

Standard format:

<type identifier> <variable>(value)

Sample implementation:

int a(0);

Strings

Strings are a data type, which is used to store non-numerical values. The length and number of values is equal to or greater than 1. The C++ standard library supports '*Strings*' data type. However, in order to use the functionality included in the library, you must include the header file, string, in the following manner.

#include<string>

Strings can be declared and defined in the following manner:

string name = "John";

An alternate method for accomplishing the same task is:

string name;

name = "John"

Constants

Expressions that contain a fixed value are called constants.

Literals

Literals are used to denote concrete values inside the source code. For instance, the integers assigned to variables like 0, in the previous example, is a literal. There are several types of literal constants, which are described in the section to follow.

Integer Numerals

They are numerical constants that distinguish between the decimal qualities of a number. Recognize that, in order to express a numerical constant, we don’t need to use any exceptional character or quotes. Whenever, we write 1871 in a system, we will be alluding to the worth 1871.

Notwithstanding decimal numbers (those that every one of us have utilized consistently), C++ permits the utilization as strict constants of octal numbers (base 8) and hexadecimal numbers (base 16). On the off chance that we need to express an octal number, we need to place a 0 (zero character) before it. Besides this, with a specific end goal to express a hexadecimal number, we need to place the characters 0x (zero, x) before the number.

Literal constants, in the same way as variables, are considered to have a particular type. Of course, number literals are of sort int. In any case, we can drive them to either unsigned form by attaching the u character to it, or long form by annexing l. In both cases, the postfix could be determined utilizing either upper or lowercase letters.

Floating Point Literals

These literals express numbers in the form of exponents and numbers. They can incorporate a decimal point, an e character (symbolic of exponent), where a number occurs after the e character. Moreover, a number may have both a decimal point and an e character.

Examples:

3.14159 / 3.14159

6.02e23 / 6.02 x 10^23

1.6e-19 / 1.6 x 10^-19

In the preceding examples, it can be seen how floating-point numbers are written. The first illustration uses decimal point whereas the last two numbers use e character to indicate a 10^x, where x is the degree of exponentiation. ‘Double’ is the default type, used for floating point representation.

Character and String Literals

Non-numerical constants also exist in C++. While a character is a single char, a string represents a string of characters. Recognize that to speak of a solitary character, we wall it in the middle of single quotes (‘). On the other hand, to express a string (which for the most part comprises of more than one character), we wall it in the middle of double quotes (“).

Boolean Literals

There are just two legitimate Boolean literals: false and true. These could be communicated in C++ as estimations of type bool by utilizing the Boolean literals false and true.

Defined Constants (#define)

You can characterize your names for constants that you utilize all the time without needing to depend on memory- expending variables, just by utilizing the #define preprocessor order.

The syntax for the same is as follows:

#define <identifier> <value>

Example:

#define EXP 10

The #define mandate is not a C++ explanation. However, it is an order for the preprocessor. In this manner, it accepts the whole line as the order and does not oblige a semicolon (;) at its end. In the event that you add a semicolon character (;) at the end, it will likewise be added in all events inside the group of the program that the preprocessor substitutes.

Declared Constants (const)

You can pronounce constants with a particular sort in the same route, as you would do with a variable, using the const keyword. Here, tabulator and pathwidth are two written constants. They are dealt with much the same as standard variables aside from the fact that their qualities can’t be altered after their definition.

Operators

When we know of the presence of variables and constants, we can start to work with them. In order to facilitate processing on these variables and constants, operators are provided by the C++ programming language. Not at all like different programming languages in which operators are basically essential words, administrators in C++ are for the most part made of signs that are not included in the letter set yet are accessible in all consoles. This makes C++ code shorter and more global, since it depends less on English words, yet obliges a tad bit of learning exertion initially.

Assignment (=)

This operator assigns a value to a variable.

Syntax:

<variable identifier> = <value>

Example:

x = 0;

This announcement relegates the number value 5 to the variable a. The part at the left of the operator (=) is known as the rvalue (right value) and the right one as the lvalue (left value). The lvalue must be a variable though the rvalue might be either a constant, a variable or any mixture of these. The most vital tenet when allocating is the right-to-left operation is that the operation dependably happens from right to left, and never the other way.

Syntax:

<lvalue> = <rvalue>;

Example:

x = y;

This announcement doles out to variable x (lvalue) the value contained in variable y (rvalue). A property that C++ has over other programming languages is that the task operation could be utilized as the rvalue (or part of a rvalue) for an alternate task operation.

Example:

x = 7 + (y = 0);

In this example statement, 0 is allocated to variable y and then add 7 to this value. The result is assigned to x.

The accompanying statement is additionally legitimate in C++:

x = y = z = 0;

It relegates 0 to the all the three variables: x, y and z.

The five arithmetical operations upheld by the C++ dialect are: add (+), subtract (-), multiply (*), division (/) and modulo (%). The stand out that you may not be so used to see is modulo. This operator gives remainder of division as the result. For example, if you perform the following operation:

a = 21 % 4;

The remainder of the division of 21 with 4 is 1. This is the output of this operation.

Compound Assignment

Compound operators include subtraction (-=), addition (+=), division (/=), multiplication (*=), >>=, <<= and modulo (%=). In addition to bitwise logical operators can also be

used. When we need to adjust the estimation of a variable by performing an operation on its present, compound assignments can be used.

Syntax:

<variable identifier><operator> = <value>

Example:

x+ = 1

In the example shown above, the value of x is incremented by 1.

Increase and Decrease (++, —)

In an attempt to reduce the number of statements used, operations increase and decrease are provided. These statements are equivalent to +=1 and to -=1, respectively.

In the early C compilers, the three past articulations presumably delivered distinctive executable code relying upon which one was utilized. These days, this sort of code streamlining, is done by the compiler, accordingly the three interpretations ought to deliver precisely the same executable code.

A normal for this operation is that it might be utilized both as a prefix and as a postfix. That implies that it could be composed either before the variable identifier (++a) or after it (a++). Albeit in basic statements like a++ or ++a, both have precisely the same importance. However, in different declarations in which the consequence of the build or abatement operation is assessed as a worth in an external interpretation, they may have a vital distinction in their significance.

In the case that the increment operation is utilized as a prefix (++a) the value is incremented before the consequence of the statement is assessed and subsequently the expanded worth is considered in the external outflow. On the off chance that that it is utilized as a postfix (a++), the value is expanded in the wake of being assessed and

accordingly the value put away before the build operation is assessed in the external representation.

Relational and Equality Operators

These operators include not equal to (!=), equality (==), lesser than (<),lesser than and equal to (<=), greater than (>) and greater than or equal to (>=).

To assess an examination between two interpretations we can utilize the relational and correspondence operations. The consequence of a relational operation is a Boolean value. This result can have one of the two values: false or true.

We may need to look at two interpretations, for instance, to know whether they are equivalent or if one is more noteworthy than the other is. Here is a rundown of the operators that could be utilized within C++:

Example:

(7 == 3) //This expression evaluates to false

(6 > 5) //This expression evaluates to true

(3 != 4) //This expression evaluates to true

(6 >= 6) //This expression evaluates to true

(5 < 5) //This expression evaluates to false

Logical Operators

C++ programming language supports three logical operations namely, not (!), and (&&) and or (||).

The Operator not (!) is the C++ administrator to perform the Boolean operation NOT. It has one and only operand, and the main thing that it does is to converse the value of it. If the value of the operand is true, then it is converted to false. On the other hand, if the

value is false, the same is converted to true. Essentially, it furnishes a proportional payback Boolean benefit of assessing its operand.

The other two operations, && and ||, make use of two variables. The operation and (&&) relates with Boolean sensible operation AND. This operation results in true if and only if both the operands are true. On the other hand, the operation || compares with Boolean operation OR and results in a true value if at least one of the two operands is true.

Conditional Operator (?)

This operator assesses a declaration giving back a value if that outflow is true and an alternate one if the statement is assessed as false.

Syntax:

conditionForTesting ? resultIfTrue : resultIfFalse

If conditionForTesting is true, the statement will return resultIfTrue. However, if the conditionForTesting is false, then it will return resultIfFalse.

Comma Operator (,)

The comma operation (,) is used on two or more independent outflows that are incorporated where stand out statement is normal. At the point when the set of declarations must be assessed for a worth, just the rightmost articulation is considered.

Bitwise Operators

Bitwise operations include bitwise and (&), bitwise or (|), ~, ^, >> and <<. These operators adjust variables considering the bit designs that speak to the qualities they store.

Explicit Type Casting Operator

This operator permits you to change over a datum of an offered type to an alternate one. There are a few approaches to do this in C++. The most straightforward one, which has

been inherited from the C dialect, is to go before the declaration to be changed over by the new type encased between enclosures (()).

Example

int i;

float f;

i = int (f);

The code mentioned above declares two variables i and f. While i is an integer, f is a float. Directly assigning f to i will invoke a warning. Therefore, you need to typecast it. The third statement converts the flat variable's value to an integer and assigns the same to the integer variable.

Operator Precedence

You may have to create complex expressions. However, this may bring forth several questions. We may have a few questions about which operand is assessed first and which later. For instance, in this representation:

x = 5 – 4 + 3;

Ideally, a left to right evaluation is performed. However, operators follow a precedence order. The following table gives the order in which operations are performed.

Operator	Level
::	1
() [] —	2

++ ->[]() typeid static_cast dynamic_cast const_cast reinterpret_cast	
++ — ~ ! sizeof new delete	3
*& +-	
(type)	4
.* ->*	5
*/%	6
+-	7
<< >>	8
< > <= >=	9

== !=	10
&	11
^	12
|	13
&&	14
||	15
?:	16
= *= /= %= += -= >>= <<= &= ^= |=	17
,	18

This table describes the priority with which operators are evaluated if two or more of the operators mentioned here are used.

Basic Input / Output In C++

Utilizing the standard include and package library, we will have the capacity to interface with the client by printing messages on the screen and getting the client's information from the console. C++ utilizes a helpful deliberation called streams to perform output and input operations in successive media, for example, the screen or the console.

A stream is an item where a system can either embed or concentrate characters to/from it. We don't generally need to think about numerous determinations about the physical media connected with the stream - we just need to know it will acknowledge or give characters successively.

The standard C++ library incorporates the header document iostream, where the standard output and input stream items are pronounced.

Standard Output (cout)

Naturally, the standard output of a project is the screen, and the C++ stream item characterized to get to it is cout. cout is utilized as a part of conjunction with the insertion operator, which is composed as <<.

Sample Implementation:

cout << "This is C++";

Standard Input (cin).

The standard data gadget is generally the console. Standard input can be used in C++ by applying the operator of extraction (>>) on the cin stream. The operator must be trailed by the variable that will store the information that is going to be concentrated from the stream.

Sample Implementation

```
int x;

cin >> x;
```

Cin and Strings

We can utilize cin to get strings with the extraction operator (>>) as we do with principal information sort variables:

```
cin >> stringIn;
```

Nonetheless, as it has been said, cin extraction quits perusing when if discovers any clear space character. So, for this situation, we will have the capacity to get only single word for every extraction.

Control Structures

A system is normally not constrained to a straight grouping of guidelines. Amid its process, it may bifurcate, rehash code or take choices. For that reason, C++ gives control structures that serve to point out what must be carried out by our system, when and under which circumstances.

A large portion of the control structures that we will see in this area, oblige a bland explanation as a major aspect of its language structure. An announcement might be either a basic articulation (a straightforward guideline finishing with a semicolon) or a compound explanation (a few directions gathered in a square). In the case that we need the announcement to be a straightforward articulation, we don't have to wall it in braces ({}). Be that as it may in the case that we need the announcement to be a compound proclamation, it must be encased between braces ({}), framing a piece.

The if-else Construct

The construct 'if' defines a condition and a set of instructions. The if-block is executed in the case that the condition is satisfied.

Syntax of the if construct is as follows:

```
if (<condition>)
{
    //Code
}
```

Here, condition is the articulation that is constantly assessed. On the off chance that this condition is true, code is executed. In the event that it is false, code is overlooked (not

executed) and the system proceeds with directly after this contingent structure.

An extended version of the if-construct is the if-else construct. The syntax of this construct is as follows:

```
if (<condition>)
{
    //Code If True
}
else
{
    //Code If False
}
```

On the off chance that this condition is true, code-if-true is executed. In the event that it is false, code-if-true is overlooked (not executed) and the system proceeds to execute code-if-false.

Example:

```
if (a == 0)
{
    cout<<"Value of a: 0";
}
else
{
    cout<<"Value of a is not 0";
```

}

Loops

Loops have as reason to rehash a piece of code a specific number of times or if the while a condition is satisfied.

Syntax of the while construct:

while (<condition>)

{

    //Code

}

Its usefulness is just to rehash articulation while the condition set in declaration is true.

Syntax of the do-while loop:

do

{

    //Code

} while (<condition>);

Its usefulness is precisely the same as the while loop, aside from the condition in the do-while loop, which is assessed after the execution of code execution rather than in the recent past, allowing no less than one execution of explanation regardless of the possibility that condition is never satisfied.

The do-while loop is generally utilized when the condition that need to focus the end of the circle is resolved inside the circle explanation itself, as in the past case, where the client enter inside the piece is what is utilized to figure out whether the circle need to end.

Syntax of the for loop:

for (<initialization>; <condition>; <build>)

{

    //Code

}

Its primary capacity is to execute code while condition stays valid, in the same way as the while loop. Anyway likewise, the for-loop gives particular areas to contain an instatement explanation and an increment/decrement proclamation. So, this loop is uniquely intended to perform a dreary activity with a counter, which is instated and expanded on every cycle.

It lives up to expectations in the accompanying way:

1. Initialization is executed

    For the most part, it is an introductory value setting for a counter variable. This is executed just once.

2. Condition is checked

    In the event that the condition is true, the loop proceeds. However, if the condition is false, the loop executes after executing the build condition.

3. Statement is executed

    Of course, it could be either a single-line code or a braces-encased lines of code like, { }.

4. Finally, whatever is determined in the build field is executed and the loop returns to step 2.

Jump Statements

C++ supports several forms of jump statements. These statements are discussed below.

The break Statement

Utilizing break, we can leave a loop regardless of the possibility that the condition for its end is not satisfied. It could be utilized to end an unending circle, or to constrain it to end before its regular end. Case in point, we are going to stop the tally down before its regular end.

The Continue Statement

The continue statement causes the code to avoid whatever is left of the loop in the current emphasis as though the end of the code piece had been arrived at, making it hop to the begin of the accompanying cycle.

The goto Statement

The goto permits to make an outright hop to an alternate point in the code. You ought to utilize this peculiarity with alert. Subsequent to its execution causes an unqualified bounce overlooking any kind of settling limitations. The goal point is distinguished by a name, which is then utilized as a contention for the goto line. A name is made of a legitimate identifier took after by a colon (:).

The exit() Function

The exit() is a function available in the cstdlib library. It can be used to end the current execution with a particular code. The syntax of using this function is as follows:

void exit (int exitcode);

The exitcode is utilized by some working frameworks and may be utilized by calling functions. By tradition, an exitcode of 0 implies that the system completed typically and some other worth implies that some lapse or startling results happened.

Sample Implementation:

```
exit (0);
```

The Switch Construct

The grammar of the switch construct is a bit particular. Its objective is to check a few conceivable values for an interpretation. Something like what we did at the start of this section with the linking of a few if and else if instructions. The syntax of this construct is:

```
switch (<integer to be monitored>)
{
        case value1:
                //code
                break;
       case value2:
                //code
                break;
        .
        .
        default:
                //code
                break;
}
```

The switch construct is a bit particular inside the C++ dialect in light of the fact that it does not use blocks. Instead, it uses labels. This strengthens us to put break code after the concerned lines of code that we need to be executed for a particular condition.

Sample Implementation:

```
int a;
cin >> a;
switch (a)
{
        case 0:
                cout <<"Entered value is 0";
                break;
       case 1:
                cout << "Entered value is 1";
                break;
        default:
                cout << "Value other than 0 and 1 entered by the user";
                break;
}
```

This code asks the user to enter an integer. If the value entered is 0, case 0 code is executed. However, if 1 entered by the user, case 1 is executed. If the user enters a random value, the switch-case skips all cases, and executes the default case. It is customary to end the default case with a break statement.

Moreover, please note that if you skip the break statements, the system executes all the cases following the true case. For instance, if you skip the break statements in the above-mentioned code and the user enters 0, the system will execute all the cases. Therefore, all

the three statements will be printed.

Functions

You can structure your code using functions and allows you to get to all the potential that organized programming can offer to you in C++. A function is a gathering of statements that is executed when it is called from some purpose of the project. The accompanying is its organization:

<data type> <function identifier> (<parameter1>, <parameter2>, …)

{

//code

}

Here,

- Data type is the data type of the information returned by the function.
- Function identifier is the identifier by which it will be conceivable to call the function.
- Parameters (the same number of as required): Each parameter comprises of a data type and identifier combination. Parameters are declared in the same manner as regular variables are declared. They permit to pass values from the calling function to the function when it is called. These parameters are, typically separated, by commas.
- Code statements form the body of the function. It is a piece of proclamations encompassed by props

Scope of Variables

The extent of variables pronounced inside a function or some other inward square is just

their function or their code-block and can’t be utilized outside of them. Consider the following example for better understanding of the scope concept.

```
void func1()
{
    int a;
    //code
}
void func2()
{
    int b;
    //code
}
```

In the sample code given above, the variable ‘a’ is available only inside func1 () where variable ‘b’ is available only inside func2 (). Hence, the extent of nearby variables is restricted to the block level in which they are declared. All things considered, we likewise have the likelihood to declare variables, which can have a global scope. These are obvious from any purpose of the code, inside and outside all functions. With a specific end goal to declare global variables, you just need to announce the variable outside any block or function. However, this location must be in the assemblage of the project.

Functions with Void Return Value

If you go back to the syntax of declaration of a function, you will see that the data type declares the data type of the value returned by the function. However, some functions may not return anything in which case you will give the data type as void.

Envision that we need to make a capacity function to demonstrate a message on the screen. We needn't bother about giving a return value. For this situation, we ought to utilize the void data type for the function. This is an uncommon specifier that shows nonattendance of type.

Void can likewise be utilized within the function's parameter rundown to unequivocally point out that we need the function to take no real parameters when it is called.

Values Passed by Reference and By Value

When calling a function with parameters, what have gone to the function were duplicates of their values. However, never the variables themselves are sent. Any changes made to the value of the variable inside the function are not reflected on the original variable. On the other hand, if a reference of the variable is sent to the function, the changes made to the variable inside the function reflect on the original variable as they are.

Default Values in Parameters

At the point when declaring a function, we can define a default value for each of the parameters. This quality will be utilized if the relating contention is left clear when calling the function. To do that, we basically need to utilize the task operator and a value for the variable in the function revelation. In the event that a value for that parameter is not passed when the function is called, the default value is utilized. However, in the event that a value is defined this default quality is disregarded and the passed value is utilized.

Overloaded Functions

In C++ two separate functions can have the same name if their parameter types or number are distinctive. That implies that you can give the same name to more than one function on the off chance that they have either an alternate number of parameters or diverse parameter data types.

A sample implementation is given below:

```
#include <iostream>

using namespace std;

float processnum (float x, float y)

{

    return (x/y);

}

int processnum (int x, int y)

{

        return (x+y);

}

int main () {

            int a=2, b=2;

            float c=4.0,m=4.0;

            cout << processnum (a, b)<<endl;

            cout << processnum (c, d)<<endl;

            return 0;

}
```

For this situation we have declared and defined two functions with the same name processnum. One of them acknowledges two parameters of sort int and the other one acknowledges them of sort float. The compiler knows which one to bring in each one case by analyzing the data passed as values when the function is called. On the off chance that

it is called with two ints as its parameters, it calls to the function that has two int parameters in its declaration. On the other hand, in the event that it is called with two floats, it will call the particular case that has two float parameters in its model.

In the first call to processnum(), the two parameters passed are of sort int. Subsequently, the function with the second declaration is called, which adds the two values and returns the result. However, the second call passes two parameters of sort float. So, the function with the first model is called. This one has an alternate conduct, which divides one parameter from the other. So, the conduct of a call to function relies on upon the sort of the parameters passed on the grounds that the function has been overloaded.

It is important to note that that a function can be overloaded in other ways as well and no less than one of its parameters must have an alternate data type.

Inline Functions

The inline specifier demonstrates to the compiler that inline substitution is desired for the concerned function. This does not change the conduct of a function itself. However, it is utilized to recommend to the compiler that the code produced by the body of the function must be embedded at the point the function is called, as opposed to being embedded just once and perform a general call to it, which for the most part includes some extra overhead in running time.

Standard Syntax:

inline <data type> <identifier> (<parameters with their data types>)

{

    //Code

}

Please note that the call is much the same as the call to any other standard function. Also,

you don’t need to incorporate the inline word while calling the function. It just needs to be specified at the declaration-definition of the function.

Recursivity

Recursivity is the property that allows calling of functions without any input or external intervention. It is valuable for implementation of some functionality such as sorting or computing the factorial of numbers.

Arrays

An array is an arrangement of components of the same data type, put in sequential memory areas that might be exclusively referenced by using the following format:

arrayRef[i]

Here, arrayRef is any array and i can be any integer value, not greater than the number of elements in the array. This implies that, for instance, we can store 5 values of int type in an array without needing to pronounce 5 separate variables. All these are referenced using one identifier. Like a normal variable, an array must be declared before it is utilized. A normal affirmation for an exhibit in C++ is:

<data type> <identifier> [<number of elements>];

Here data type is the data type of the elements that the array will hold, identifier is a unique name given to the array and number of elements indicate the number of elements present in the array.

Example:

If you want to store five numbers of integer type, you need to declare this structure using the following declaration:

int num [5];

Initializing Arrays

At the point when proclaiming a customary exhibit of array (inside a function, for instance), in the event that we don’t tag else, its components won’t be instated to any value naturally. So, their substance will be undetermined until we store some value in them. The components of global and static nature, then again, are naturally instated with their default

values. Therefore, all the elements of global or static arrays are initialized to 0.

Accessing Elements Of The Array

In any purpose of a system in which an array is obvious, we can get to the value of any of its element independently as though it was a typical variable, accordingly having the capacity to both read and adjust its value. The configuration is as straightforward as:

<identifier> [<index>]

For example, if you wish to access the first element of array num, declared in the previous section, you must use the statement:

a = num [0];

In C++, it is linguistically right to surpass the substantial scope of records for a array. This can create issues, since getting to out-of-reach components don’t result in gathering blunders yet can result in runtime slips. The motivation behind why this is permitted will be seen further ahead when we start to utilize pointers.

Multidimensional Arrays

Multidimensional arrays might be depicted as “array of arrays”. Case in point, a bi-dimensional array might be envisioned as a bi-dimensional table made of components, every one of them of a same uniform information sort.

Multidimensional clusters are not restricted to two indexes (i.e., two measurements). They can contain the same number of records as required. However be watchful! The measure of memory required for a cluster quickly increments with each one increment.

Arrays as Parameters

At some point, we may need to pass an array to a function as a parameter. In C++, it is impractical to pass a complete square of memory by value as a parameter to a function. However, we are permitted to pass its address. In practice, this has very nearly the same

impact and it is a much speedier and a more productive operation.

Keeping in mind the end goal to acknowledge arrays as parameters, the main thing that we need to do when declaring the function is to point out in its parameters the component kind of the array, an identifier and a couple of void sections [].

Character Sequences

As you may know, the C++ Standard Library executes an effective string class, which is extremely helpful to handle and control series of characters. On the other hand, on the grounds that strings are indeed arrangements of characters, we can speak to them likewise as plain arrays of char components.

Initialization of Null-Terminated Character Sequences

Since clusters of characters are normal arrays, they take after all their same standards. For instance, in the event that we need to introduce an array of characters with some foreordained arrangement of characters, we can do it much the same as any other. A sample implementation of this functionality is:

char name[] = { ‘J’, ‘o’, ‘h’, ‘n’, ‘\0’ };

In this case, you could have declared a char array with 5 elements. While you can initialize char arrays in the manner shown above, you can perform the same function in a better manner using string literals.

Using null-terminated sequences of characters

The standard way of defining strings in C++ programming language is as a sequence of characters that are terminated with a null character (\0). This is the format used for string literals as well.

Pointers

We have as of now perceived how variables are seen as memory cells that could be gotten to utilizing their identifiers. Thusly, we didn’t need to think about the physical area of our information inside memory. We basically utilized its identifier at whatever point we needed to allude to our variable.

The memory of your machine could be envisioned as a progression of memory cells, every one of the negligible sizes, which machines deal with (one byte). The memory cells are sequential in nature. Therefore, every element in this block is the same number as the past one in addition to one.

Reference Operator (&)

When we declare a variable, the measure of memory required is allocated for it at a particular area in memory (its memory address). We, by and large, don’t heartily choose the accurate area of the variable inside the board of cells that we have envisioned the memory to be.

Fortunately, that is an undertaking naturally performed by the working framework at runtime. Be that as it may, sometimes, we may be intrigued about knowing the location where our variable is constantly put away during runtime so as to work with relative positions to it.

The address that spots a variable inside memory is the thing that we call a reference to that variable. This reference to a variable might be gotten by placing before the identifier of a variable an ampersand sign (&), known as reference operator, and which could be actually interpreted as “location of”.

Dereference Operator (*)

We have quite recently seen that a variable, which stores a reference to an alternate variable, is known as a pointer. Utilizing a pointer, we can specifically get to the value put away in the variable, which it indicates. To do this, we basically need to place before the pointer's identifier a reference bullet (*), which goes about as dereference operator and that could be actually mean "value pointed by".

Contrast Between Reference and Dereference Operators

- Ampersand (&) is the reference operator and might be perused as "location of"
- Asterisk (*) is the dereference operator and might be perused as "quality pointed by"

Pointers and Arrays

The idea of array is really bound to the one of pointer. Actually, the identifier of a cluster is equal to the location of its first component, as a pointer is identical to the location of the first component that it indicates. Truth be told, they are the same idea.

The accompanying task operation would be legitimate:

p = num;

If p is a pointer and num is an array, both of the same type, p and numbers would be comparable and would have the same properties, after this statement. The main contrast is that we could change the value of pointer p by another, while numbers will dependably indicate the first of the different elements of a fixed data type with which it was characterized.

Hence, dissimilar to p, which is a conventional pointer, numbers is a array. Therefore, an array could be viewed as a consistent pointer. In this manner, the accompanying designation would not be substantial:

num = p;

Since numbers is an array, it works as a steady pointer, and we can’t relegate values to constants.

Pointers to Pointers

C++ permits the utilization of pointers that indicate pointers, which in its turn, point to information (or even to different pointers). With a specific end goal to do that, we just need to include a bullet (*) for each one level of reference in their revelations.

Null Pointer

A null pointer is a customary pointer of any pointer, which has an exceptional value that demonstrates that it is not indicating any legitimate reference or memory address. This quality is the consequence of sort throwing the number value zero to any pointer sort.

Don’t mistake null pointers for void pointers. A null pointer is a pointer that does not point to anything. On the other hand, a void pointer points to a location that contains a value of void type. One alludes to the value put away in the pointer itself and the other to the sort of information it indicates.

Pointers to Functions

C++ permits operations with pointers to functions. The common utilization of this is for passing a capacity as a parameter to an alternate function, since these can’t be passed dereferenced. To declare a pointer to a capacity, we need to announce it like the declaration of the function aside from that the name of the function is encased between enclosures () and a mark (*) is embedded before the name.

Dynamic Memory Allocation

As of recently, in all our examples, we had knew about the amount of memory we needed and we declared our variables, even arrays, on the basis of this estimate. As a result, we had all the memory required by the program allocated well in advance, before the execution of the code began.

At the same time, suppose you are faced with a situation where you require a variable measure of memory and this measure must be resolved amid runtime. For instance, in the case you require some client data to determine the essential measure of memory space required.

The answer to such a requirement is dynamic memory, for which C++ allows the operators new and delete.

Operators new and new []

With a specific end goal to ask for element memory, we utilize the operator new. A data type specifier trails the operator ‘new’. However, if a grouping of more than one component is needed, the quantity of these can be mentioned inside sections []. It gives back a pointer to the start of the new square of memory distributed.

Syntax:

<pointer identifier> = new <data type>

<pointer identifier = new <type> [<number_of_elements>]

The main statement is utilized to dispense memory to contain one single component of the specified data type. The second one is utilized to allot an array of components of the specified data type, where number_of_elements is a whole number value speaking to the

measure of these.

Operators delete and delete[]

Since the need of element memory is generally constrained to particular minutes inside a code, once it is no more required, it ought to be liberated so that the memory gets to be accessible again for different solicitations of element memory. This is the reason why the operator delete is provided.

Syntax:

delete <pointer identifier>;

delete [] <pointer identifier>;

The main representation ought to be utilized to erase memory apportioned for a solitary component, and the second one for memory allotted for array of components. The value passed as parameter to erase must be either a pointer to a memory apportioned with new, or a null pointer (on account of an null pointer, delete creates no impact).

Introduction to Object Oriented Programming: Classes

A class is an extended idea of an information structure, as opposed to holding just information. It can hold both functions and data. An object, on the other hand, is an instantiation of a class. Regarding variables, a class would be the type, and an object would be the variable.

Classes are by and large declared utilizing the keyword class, with the accompanying syntax:

class <className> {

accessSpecifier_1:

    memberData;

    memberFunction;

access_specifier_2:

    memberData;

    memberFunction;

} <objectName>;

Where className is an identifier for the class, objectName is a nonobligatory identifier of names for objects of this class. The assortment of the presentation can contain parts, which could be either data members or function statements, and alternatively get to specifiers.

Example:

int a;

Here, ‘a’ is an object of the int class.

Access Specifiers

All is fundamentally the same to the statement on information structures, aside from that we can now incorporate additionally functions and data parts, and this new thing called access specifier. A right to gain access or access specifier includes one of the accompanying three keywords: private, public or protected. These specifiers alter the right to gain entrance access to the members that fall under its scope.

- Private:

  Such members of a class are available just from inside different members of the same class or from their companions.

- Protected

  Such members are available from members of their same class and from their companions, additionally from parts of their inferred classes.

- Public

  Such members are available from anyplace where the item is noticeable.

By default, all members of a class are private to access for all its members.

Example:

```
class myRectangle {
        double a, b;
        public:
        void setValues (double, double);
        double func (void);
} myRect;
```

In this example, a and b are private while functions setValues() and func() are public.

Constructors and Destructors

You, for the most part, need to introduce variables or allocate dynamic memory during the methodology of creation to clear up memory and to abstain from returning startling values during the execution of the program.

If you consider the previous example, had you sent ‘a’ and ‘b’ as parameters to a function without initializing them, the values taken by the function will be garbage. Your code will most likely give wrong results or crash abruptly.

Keeping in mind the end goal to keep away from that, a class can incorporate a unique function called constructor, which is naturally called at whatever point another object of this class is made. This constructor function must have the same name as the class, and can’t have any return data type.

Example:

```
#include <iostream>

using namespace std;

class myRectangle {

        double x, y;

    public:

        myRectangle (double, double);

        double areaRect () {return (x*y);}

};

myRectangle :: myRectangle (double a, double b) {

        x = a;

        y = b;
```

```
}
int main ()
{
        myRectangle rectx (1,2);
        cout << “rect area: ” << rectx.areaRect() << endl;
        return 0;
}
```

The first statement in the main() instantiates the class myRectangle. The parameters passed with the creation of the object called the constructor function and initialize the dimensions of the rectangle to 1 and 2. Therefore, when the areaRect() is called, the function returns the correct value of area, 2.

Constructors can’t be called unequivocally as though they were normal functions. They are just executed when another object of that class is made. You can likewise perceive how not one or the other, the constructor function (inside the class) nor the last constructor definition incorporate a return value, not even void.

The destructor satisfies the inverse usefulness. It is consequently called when an item needs to be demolished, either on the grounds that its extent of presence has completed (for instance, on the off chance that it was characterized as a nearby protest inside a function and the function closes) or in light of the fact that it is an article alertly appointed and it is discharged utilizing the operator delete.

The destructor must have the same name as the class. However, a tilde sign (~) is placed before the name and it should likewise give back no return value. The utilization of destructors is particularly suitable when an object uses dynamic memory amid its lifetime and right now of being annihilated, we need to discharge the memory that the item was

designated.

Overloading Constructors

Like other functions, a constructor can likewise be overloaded with more than one function definitions that have the same name yet distinctive sorts or number of parameters. Keep in mind that for overloading to work, the compiler will call the one whose parameters match the values utilized as a part of the function call. On account of constructors, which are consequently called when an object is made, the one executed is the particular case that matches the values passed on the item assertion.

Default Constructor

In the event that you don’t declare any constructors in a class definition, the compiler accepts the class to have a default constructor with no parameters. The compiler not just makes a default constructor for you on the off chance that you don’t create your own.

It gives three extraordinary functions altogether that are certainly declared in the event that you don’t declare your own. These are the copy constructor, the default destructor and the copy assignment operator.

The copy constructor and the copy assignment operator duplicate all the information contained in an alternate object to the members of the current object.

Pointers to Classes

It is flawlessly legitimate to make pointers that indicate classes. We essentially need to consider that once declared, a class turns into a substantial data type, so we can utilize the class name as the data type for the pointer.

Example:

myRectangle * pointerRect;

The pointerRect is a pointer to an object of class myRectangle.

As it happened with information structures, so as to allude specifically to a specific member of an object pointed by a pointer, we can utilize the arrow operator (->) of indirection.

Overloading Operators

C++ consolidates the alternative to utilize standard operators to perform operations with classes notwithstanding with major data types. C++ has a functionality that allows you to overload operators by planning classes ready to perform operations utilizing standard administrators.

To over-burden an operator with a specific end goal to utilize it with classes we proclaim operator functions, which are consistent functions whose names are the function keywords emulated by the operator sign that we need to overload. The arrangement is:

<data type> <operator> <operator sign> (<parameters>)

{

//Code

}

Here you have an illustration that overloads the addition operator (+). We are going to make a class to store 2-D vectors and afterward, we are going to include two of them: a (2,4) and b (3,4). The addition of two 2-D vectors is an operation as basic as adding the two x directions to get the ensuing x direction. The result of this operation on the points ‘a’ and ‘a’ should be (2+3, 4+4), which is equal to (5, 8).

#include <iostream>

using namespace std;

class myVector {

public:

int a, b;

```
        myVector () {};
        myVector (int, int);
        myVector operator + (myVector);
};
myVector :: myVector (int x, int y) {
         a = x;
        b = y;
       }
        myVector myVector :: operator+ (myVector myParam) { myVector tempo;
        tempo.a = a + myParam.a;
        tempo.b = b + param.b;
        return (tempo);
}
int main () {
         myVector x (2, 4);
         myVector y (3, 4);
         myVector z;
         z = x + y;
         cout << z.a << “,” << z.b;
         return 0;
}
```

Keyword this

The word this speaks of a pointer to the object whose member function is continuously executed. It is a pointer to the object itself.

It is additionally regularly utilized as a part of operator= function that returns objects by reference (keeping away from the utilization of impermanent objects). Taking after with the vector's illustrations seen before we could have composed an operator= capacity like this one.

Actually this capacity is very much alike to the code that the compiler creates verifiably for this class on the off chance that we do exclude an operator= part function to duplicate objects of this class.

Static Members

A class can contain static parts, either data or functions. Static data of a class are otherwise called "class variables", on the grounds that there is stand out novel values for all the objects of that same class. Their substance is not the same as one object of this class to an alternate.

Friend Functions

On a fundamental level, private members of a class can't be gotten to from outside the same class in which they are pronounced. Then again, this standard does not influence friend functions.

On the off chance that we need to proclaim an outside function as friend of a class, in this manner permitting this capacity to have admittance to the private and secured members of this class, we do it by proclaiming a model of this outer capacity inside the class, and placing the keyword friend before it.

Example:

```
#include <iostream>
using namespace std;
class myRectangle {
	int x, y;
	public:
		void setValues (int, int);
		int func () {return (x * y);} friend myRectangle copy (myRectangle);
};
void myRectangle :: setValues (int a, int b) {
	x = a;
	y = b;
}
myRectangle copy (myRectangle rectParam) {
	myRectangle rects;
	rects.x = rectParam.x*2;
	rects.y = rectParam.y*2;
	return (rects);
}
int main () {
	myRectangle rectx, recty;
	rectx.setValues (2,3);
```

```
    recty = copy (rectx);
    cout << recty.func();
    return 0;
}
```

The copy function is a friend function of myRectangle. From inside that function, we have possessed the capacity to get to the private members of the class, ‘x’ and ‘y’. Recognize that not one or the other in the statement of copy() nor in its later use in principle() have we considered copy a part of class myRectangle. It isn’t! It essentially has entry to its private and secured members without being a part of the class.

The friend function can serve, for instance, to direct operations between two separate classes. For the most part, the utilization of friend function is out of an item arranged programming technique, so at whatever point conceivable it is better to utilize parts of the same class to perform operations with them.

Friend Classes

Generally as we have the likelihood to declare a friend function capacity, we can likewise declare a class as friend of another, allowing that top of the line access to the ensured and private members of the second one.

Example

```
#include <iostream>
using namespace std;
class mySquare;
class myRectangle {
    int x, y;
```

```
        public:
        int myArea () {return (x * y);}
        void convert (mySquare a);
};
class mySquare {
      private:
            int x;
        public:
             void setSide (int a)
             {side=a;}
             friend class myRectangle;
};
void myRectangle::convert (mySquare a) {
x = a.x;
y = a.x;
}
int main () {
        mySquare sqr;
        myRectangle rect;
        sqr.setSide(4);
        rect.convert(sqr);
```

```
    cout << rect.myArea();

    return 0;

}
```

In this illustration, we have declared myRectangle as a companion of mySquare so that myRectangle functions could have entry to the ensured and private members of mySquare, all the more solidly to Csquare::x, which portrays the side of the square.

Inheritance between Classes

A key feature of C++ classes is inheritance, which permits you to make classes that are inferred from different classes. So, they naturally incorporate some of its “parent’s” members, in addition to its own.

Classes that are inferred from others inherit all the available parts of the base class. That implies that if a base class incorporates a part A and we determine it to an alternate class with an alternate part called B, the determined class will contain both parts A and B.

To determine a class from an alternate, we utilize a colon (:) as a part of the revelation of the inferred class utilizing the accompanying arrangement:

class <derived class name>: public <base_class_name>

{

    //Code

};

Where derived_class_name is the name of the inferred class and base_class_name is the name of the class on which it is based. The public specifier may be supplanted by any of alternate access specifiers protected and private. This access specifier portrays the base access level for the parts that are inherited from the base class.

#include <iostream>

using namespace std;

class myPol {

        protected:

```
        double x, y;
    public:
        void setValues (double a, double b)
        { x = a; y = b;}
};
class myRect: public myPol {
     public:
     double myArea ()
     {
        return (x * y);
     }
};
class myTri: public myPol {
    public:
    int myArea ()
    {
        return (x * y/ 2);
    }
};
int main () {
     myRect rect;
```

```
    myTri trgl;
    rect.setValues (4,5);
    trgl.setValues (4,5);
    cout << rect.myArea() << endl;
    cout << trgl.myArea() << endl;
    return 0;
}
```

The objects of the classes, myRectangle and myTriangle, each one contain members inherited from myPolygon. These are: x, y and setValues().

The protected access specifier is like private. Its contrast happens actually with inheritance. At the point when a class inherits from another, the parts of the inherited class can get access to the protected members inherited from the base class, however not its private members.

Since we needed x and y to be public from parts of the inferred classes myRectangle and myTriangle and not just by parts of myPolygon, we have utilized protected rather than private.

What Is Inherited From The Base Class

On a basic level, an inherited class inherits each member of a base class with the exception of:

- Constructor
- Destructor
- Operator=() members
- Friends

Although, the destructors and constructors of the base class are not inherited themselves, the destructor and default constructor are constantly called when another object of an inherited class is made or destroyed. In the event that the base class has no default constructor or you need that an overload constructor is called when another inherited object is made, you can detail it in every constructor meaning of the determined class:

<derived constructor name> (<parameters>) : <base constructor name> (<parameters>) {

    //Code

}

Multiple Inheritance

There is a provision for a class to inherit from multiple classes in C++. This can be carried out, by separating the base classes, with commas in the class declaration. Case in point, in the event that we had a particular class to print on screen (myOutput) and we needed our classes myRectangle and myTriangle to likewise inherit its members notwithstanding those of myPolygon we could compose them accordingly.

Implementing Polymorphism

Before getting into this segment, it is prescribed that you have a legitimate understanding of pointers and class inheritance. In the event that any of the accompanying explanations appear unusual to you, you ought to audit the showed segments:

Pointers to base class

One of the key peculiarities of inherited classes is that a pointer to an inferred class is sort good with a pointer to its base class. Polymorphism is the craft of exploiting this straightforward however compelling and flexible peculiarity that brings Object Oriented Methodologies to its maximum capacity.

Virtual Members

A part of a class that could be reclassified in its inferred classes is known as a virtual member. With a specific end goal to declare a part of a class as virtual, we must place before its statement the keyword virtual.

What the virtual key word does is that it permits a part of a determined class with the same name as one in the base class to be properly called from a pointer, and all the more accurately when the kind of the pointer is a pointer to the base class yet is indicating an object of the inferred class.

Abstract Base Class

The fundamental distinction between a theoretical base class and a customary polymorphic class is that in light of the fact that in conceptual base classes no less than one of its parts needs usage, we can’t make cases (items) of it. Be that as it may a class that can’t instantiate objects is not completely futile. We can make pointers to it and exploit all its polymorphic capabilities.

Virtual members and dynamic classes stipend C++ the polymorphic qualities that make item situated programming such a helpful instrument in huge activities. Obviously, we have seen exceptionally basic employments of these features, however these peculiarities could be connected to exhibits of items or progressively apportioned objects.

Function Templates and Class Templates

Function templates are unique functions that can work with non-specific data types. This permits us to make a function layout whose usefulness could be adjusted to more than one sort or class without rehashing the whole code for each one sort.

In C++, this could be attained utilizing template parameters. A template parameter is an uncommon sort of parameter that might be utilized to pass a type as parameter: much the same as normal parameters could be utilized to pass values to a function, format parameters permit to pass likewise sorts to a function. These templates can utilize these parameters as though they were some other customary data type.

The syntax used for this is as follows:

template <typename identifier> function_declaration;

template <class identifier> function_declaration;

Both the statements written above have the same meaning and effect. The only difference lies in the use of keywords, typename or class.

Example:

To illustrate the concept of function templates, let us create a template that returns the higher value of the two parameters provided to it.

```
template <class eType>
eType GetMaxVal (eType x, eType y) {
        return (x>y?x:y);
}
```

int a,b;

GetMaxVal <int> (a,b);

In this example, a template parameter, eType has been used, which is symbolic of the data type that has not be specified as yet. This parameter can be used as a data type for the function just like any other data type. The GetMaxVal function template returns the greater value of the two parameters provided to it. In order to call this function template, you need to make the following function call.

<function name> <data type> (<parameters>);

For example, if you wish to call GetMaxVal for int, then you can make the following call:

GetMaxVal <int>(a, b)

Here, ‘a’ and ‘b’ are any two integers.

Sample Code:

```
#include <iostream>

using namespace std;

template <class X>

X GetMaxVal (X a, X b)

{

        X val;

        result = (a>b)? a : b;

        return (result);

}

int main () {
```

```
    int x = 1, y = 2, z;
    long i=21, j=10, k;
    z=GetMaxVal<int>(x,y);
    k=GetMaxVal<long>(i,j);
    cout << z << endl;
    cout << k << endl;
    return 0;
}
```

In this example, X is used as a data type. However, you can use a name you like. The two function calls to GetMaxVal are for different data types, int and long. It has, in fact, being used as a data type for creating new objects as well. The output of the function is the of the data type for which the function is being called.

In the above example, the compiler will implicitly call the template function with int if you call the function with parameters ‘x’ and ‘y’. However, if you call the function with long parameters, the compiler will call the template function for long. There is no need to specify <int> or <long> in these cases as the parameters used for calling the template function have been declared already. Therefore, the following function call shall suffice:

```
GetMaxVal (x, y);
```

However, please note that you cannot use parameters of different types in the function call. If you need to do something of this sort, then you must define two data types like X in the template function.

Class templates

Just like function templates, you can also choose to write class templates. The members of

this class can use the user-defined type as parameters.

```
template <class myType>
class myNums {
    myType vals [2];
    public:
      myNums (myType one, myType two)
      {
                vals[0]=first; vals[1]=second; }
    };
```

Class templates use members with data types, which are defined by the user. The template class illustrated above creates an object, which can store two numbers. These two numbers can be of any data type. The user-defined data type myType is used to indicate this type. The class template can be used for creating object pairs of different types in the following manner:

```
myNums<int> myintnum (334, 45);
myNums<double> myintfloat (1.0, 4.25);
```

Sample Code:

```
#include <iostream>
using namespace std;
template <class myType>
class myNum {
        myType x, y;
```

```
    public:
    myNums (myType one, myType two)
    {x=one; y=two;}
    myType getMaxVal ();
};
template <class myType>
myType myNums<myType>::getMaxVal ()
{
        T retValue;
    retValue = x>y? x : y;
    return retValue;
}
int main () {
        myNums <int> myOb (125, 27);
        cout << myOb.getMaxVal();
        return 0;
}
```

Namespaces

Namespaces permit you to club elements like classes, functions and objects under a name. Thusly the worldwide extension could be separated in “sub-scopes”, every unified with its name.

Syntax for namespace:

namespace <identifier name>

{

    //elements

}

Where identifier is any substantial identifier and elements is the situated of classes, functions and objects that are incorporated inside the namespace. The usefulness of namespaces is particularly helpful in the case that there is plausibility that a global function or object utilizes the same identifier as another, bringing about redefinition blunders.

Example:

namespace eNamespace

{

    int x, y;

}

eNamespace::x

eNamespace::y

```
#include <iostream>

using namespace std;

namespace sample1

{

        double myvar = 5;

}

namespace sample2

{

        double myvar = 3.1416;

}

int main () {

          cout << sample1::var;

          cout<<"\n";

          cout << sample2::var;

          cout<<"\n";

          return 0;

}
```

For this situation, there are two variables with the same name: myvar. One is defined inside the namespace sample1 and the other one in sample2. No redefinition blunders happen because of namespaces.

The using Keyword

The word using is utilized to present a name from a namespace to interfere with another of

the same name in another namespace.

Example:

```
#include <iostream>

using namespace std;

namespace sample1

{

        double x = 0.1;

        double y = 1.5;

}

namespace sample2

{

        double x = 1.1123;

        double y = 5.645;

}

int main () {

        using sample1::x;

        using sample2::y;

        cout << x;

        cout<<"\n";

        cout << y;

        cout<<"\n";
```

```
    cout << sample1::y;
    cout<<"\n";
    cout << sample2::x;
    cout<<"\n";
    return 0;
}
```

Recognize how in this code, x (without any name qualifier) alludes to sample1::x while y alludes to sample2::y, precisely as our utilizing assertions have defined. In any case we have admittance to sample1::y and sample2::x utilizing their completely qualified names. The keyword can likewise be utilized as an order to present a whole namespace:

Example:

```
#include <iostream>
using namespace std;
namespace sample1
{
    int x = 56;
    int y = 70;
}
namespace sample2
{
    double x = 1.1125;
    double y = 5.5567;
```

```
}
int main () {
        using namespace sample1;
        cout << x;
        cout<<"\n";
        cout << y;
        cout<<"\n";
        cout << sample2::x;
        cout<<"\n";
        cout << sample2::y;
        cout<<"\n";
        return 0;
}
```

This will print the sample1 values followed by the sample2 values.

Namespaces are valid only within the blocks that they are declared. Case in point, in the event that we had the proposition to first utilize the objects of one namespace and afterward those of another, we could do something like:

Example:

```
#include <iostream>
using namespace std;
namespace sample1
{
```

```
        double x = 0;
}
namespace sample2
{
        double x = 1.1216;
}
int main () {
        {
                using namespace sample1;
                cout << x
                cout<<"\n";
        }
        {
                 using namespace sample2;
                 cout << x
                 cout<<"\n";
        }
return 0;
}
```

This code will print the x value for sample1 and then print the x value for sample2.

Namespace Aliasing

We can declare alias names for existing namespaces as per the accompanying configuration:

namespace newName = presentName;

Namespace std

All the documents in the C++ standard library proclaim every last bit of its elements inside the std namespace. That is the reason why all programs have the statement that specifies that the code is using namespace ‘std ‘at the beginning of the program.

Exceptions Handling

Exception handling gives an approach to respond to extraordinary circumstances (like runtime blunders) in our system by exchanging control to unique capacities called handlers. However, in order to handle exceptions, you are required to put your code in the exception inspection.

At the point when an uncommon situation emerges inside that block, an exemption is tossed that exchanges the control to the special case handler. In the event that no special case is tossed, the code proceeds with regularly and all handlers are overlooked.

An exception is thrown by utilizing the throw keyword from inside the attempt piece. Handlers are declared with the keyword catch, which must be put promptly after the try block:

```
#include <iostream>

using namespace std;

int main () {

try

        {

                throw 0;

        }

        catch (int e)

        {

                cout << “Catch Block Entered!”
```

```
        cout << “Exception ID:” << e;
        cout<<”\n”;
    }
    return 0;
}
```

This code output the statement in the cout along with the exception ID.

The exception handler is announced with the catch keyword. As should be obvious, it starts after the end of the try square. The catch arrangement is like a consistent function that dependably has no less than one parameter. The sort of this parameter is extremely imperative, since the kind of the contention passed by the throw outflow is checked against it, and just in the case they match, the exemption is gotten.

We can chain different handlers (try outflows), every unified with an alternate parameter sort. Just the handler that matches its write with the contention tagged in the toss proclamation is executed.

After a special case has been taken care of the project execution continues after the try-catch, not after the throw!

Specifications of Exceptions

At the point when declaring a function, we can restrict the exception type. It may specifically or by implication throw by affixing a throw addition to the function declaration.

```
float myFunction (char myParam) throw (int);
```

This declares a function myFunction, which takes one argument and returns a float. The main exception that this capacity may throw is a special case of int. In the event that it

throws a special case with an alternate sort, either specifically or in a roundabout way, then an int-type handler cannot catch the same.

In the event that this throw specifier is left vacant with no type, this implies the function is not permitted to throw exceptions. Functions with no specifier are permitted to throw special cases of any type.

Standard Exceptions

The C++ Standard library gives a base class particularly intended to declare objects that are thrown as exceptions. These are present in the header file, <exception>.

It is prescribed to incorporate all dynamic memory designations inside a try obstruct that gets this kind of exception to perform a cleaning activity rather than an irregular end, which is the thing that happens when this type of special case is thrown and not caught.

Typecasting

Changing over an interpretation of a given data type into an alternate data type is known as type casting. We have as of now seen a few approaches to type cast.

Implicit Conversion

Certain changes don’t require any operator. They are naturally performed when a value is replicated to another sort. The value of ‘a’ has been elevated from short to int and we have not needed to determine any data type casting operator. This is known as a implicit change.

Implicit changes influence fundamental data types, and permit transformations, for example, the transformations between numerical types (short to int, int to float) and some pointer changes. Some of these changes may intimate a loss of exactness, which the compiler can motion with a cautioning. This could be kept away from with an express transformation.

Implied transformations likewise incorporate constructor or operator changes, which influence classes that incorporate particular constructors or administrator capacities to perform transformations.

Explicit Conversion

C++ is a solid programming language. Numerous transformations, exceptionally those that suggest an alternate elucidation of the value, require an explicit transformation. We have as of now seen two documentations for this type of typecasting namely, c-like and functional.

The usefulness of these conversions is sufficient for most needs with essential data types. Notwithstanding, these operators might be connected randomly on classes and pointers to

classes, which can prompt code that while being linguistically right can result in runtime blunders.

Conventional typecasting permits to change over any pointer into whatever other pointer sort, freely of the sorts they indicate. The ensuing call to a member come about will create either a run-time lapse or a surprising result.

Keeping in mind the end goal to control these sorts of transformations between classes, we have four particular operators: reinterpret_cast, dynamic_cast, const_cast and static_cast. Their functionality is explained below.

dynamic_cast

dynamic_cast might be utilized just with pointers and references to protests. Its design is to guarantee that the aftereffect of the change is a legitimate complete object of the asked for the class. Therefore, dynamic_cast is effective when we cast a class to one of its base classes.

static_cast

static_cast can perform transformations between pointers and related classes, not just from the inferred class to its base, additionally from a base class to its determined. This guarantees that at any rate the classes are perfect if the correct object is changed over. However, no wellbeing check is performed amid runtime to check if the item being changed over is right.

Along these lines, it is dependent upon the developer to guarantee that the change is sheltered. On the other side, the overhead of the sort security checks of dynamic_cast is dodged. static_cast can likewise be utilized to perform whatever other non-pointer change that could be performed verifiably, as for instance standard transformation between fundamental data types.

reinterpret_cast

reinterpret_cast changes over any pointer type to whatever other pointer type, even of random classes. The operation result is a basic double duplicate of the value from one pointer to the next. All pointer transformations are permitted: not the element pointed or the pointer sort itself is checked.

It can likewise cast pointers to or from whole number sorts. The configuration in which this whole number value speaks to a pointer is stage particular. The main surety is that a pointer cast to a number sort expansive enough to completely contain it, is conceded to have the capacity to be thrown over to a substantial pointer.

The changes that might be performed by reinterpret_cast, however not by static_cast have no particular uses in C++ are low-level operations, whose understanding brings about code which is by and large framework particular, and subsequently non-versatile.

This is substantial C++ code, despite the fact that it doesn’t bode well, since now we have a pointer that indicates an object of a contradictory class, and accordingly dereferencing it is risky.

const_cast

This kind of throwing controls the constant-ness of an object, either to be set or to be uprooted.

typeid

typeid permits to check the kind of an outflow: typeid (interpretation)

This operator gives back a reference to a object of sort type_info that is characterized in the standard header record <typeinfo>. This returned quality might be contrasted and another utilizing administrators == and != or can serve to acquire an invalid ended character succession speaking to the information sort or class name by utilizing its name()

part.

At the point when typeid is connected to classes, typeid utilizes the RTTI to stay informed regarding the type of element items. At the point when typeid is connected to a code whose type is a polymorphic class, the result is the kind of the most determined complete object.

Preprocessor directives

The lines included in the code of our example that are not program code yet mandates for the preprocessor are called processor directives. These lines are constantly used and can be identified with a hash sign (#). The preprocessor is executed before the genuine assemblage of code starts. In this way, the preprocessor processes all these mandates before any code is created by the announcements.

These preprocessor mandates expand just over a solitary line of code. When a newline character is found, the preprocessor mandate is considered to end. No semicolon (;) is normally used at the end of a preprocessor mandate. The main way a preprocessor order can stretch out through more than one line is by going before the newline character at the end of the line by an oblique punctuation line (\).

Macro definitions

To characterize preprocessor macros we can utilize #define. Its syntax is:

#define <identifier name> <substitution code>

At the point when the preprocessor finds this directive, it replaces any event of identifier in whatever remains of the code by substitution. This substitution might be a code, an announcement, a piece or just anything. The preprocessor does not comprehend C++. It basically replaces any event of identifier by substitution.

Example:

#define getMaxVal(x,y) x>y?x:y

This would supplant any event of getMaxVal and its two parameters, by the substitution interpretation, additionally supplanting every contention by its identifier, precisely as you

would expect on the off chance that it was a function.

Macros are not influenced by structure. A macro endures until it is unclear with the #undef preprocessor order. Function macro definitions acknowledge two unique operators (# and ##) in the substitution sequence. If the administrator # is utilized before a parameter is utilized as a part of the substitution arrangement, that parameter is supplanted by a string strict (as though it were encased between quotes)

Conditional Directives

These mandates permit to incorporate or toss some piece of the code of a project if a certain condition is met.

#ifdef permits a segment of a project to be aggregated just if the macro that is

#ifndef serves for the precise inverse: the code in the middle of #ifndef and #endif mandates is just arranged if the indicated identifier has not been long ago characterized.

The #if, #else and #elif (i.e., “else if”) orders serve to indicate some condition to be met in place for the bit of code they encompass to be gathered. The condition that takes after #if or #elif can just assess steady statements, including macro representations.

The entire structure of #if, #elif and #else binded mandates closes with #endif.

Line Control (#line)

When we direct a system and some blunder happen amid the accumulating process, the compiler demonstrates a lapse message with references to the name of the record where the mistake happened and a line number, so it is simpler to discover the code producing the slip.

The #line order permits us to control both things, the line numbers inside the code documents and the record name that we need that shows up when a lapse happens. Its syntax is:

#line number “filename”

Here, number is the new line number that will be doled out to the following code line. The line number of progressive lines will be expanded one by one starting here on.

“filename” is a nonobligatory parameter that permits to reclassify the document name that will be demonstrated. For instance:

File Inclusions

This mandate has likewise been utilized diligently as a part of different areas of this exercise. At the point when the preprocessor finds a #include order, it replaces it by the whole content of the defined record. There are two approaches to determine a record to be incorporated.

The main contrast between both representations is the spots (indexes) where the compiler is going to search for the record. In the first situation, where the document name is tagged between quotes, the record is looked first in the same catalog that incorporates the record containing the mandate. On the off chance that that it is not there, the compiler seeks the document in the default indexes where it is arranged to search for the standard header records.

On the off chance that the document name is encased between point sections <>, the record is sought straightforwardly where the compiler is arranged to search for the standard header documents. Consequently, standard header records are normally included in edge sections, while other particular header documents are incorporated utilizing quotes.

Pragma Directives

This order is utilized to define assorted alternatives to the compiler. These alternatives are particular for the stage and the compiler you utilization. Look through the manual or the

reference of your compiler for more data on the conceivable parameters that you can characterize with #pragma. On the off chance that the compiler does not help a particular contention for #pragma, it is disregarded - no slip is created.

C++ Standard Library – Input / Output with files

C++ gives the accompanying classes to perform output and input of characters to/from records:

- ofstream: Stream class to edit on documents
- ifstream: Stream class to read from documents
- fstream: Stream class to both read and edit from/to documents.

These classes are inferred straightforwardly or in a roundabout way from the classes, ostream and istream. We have effectively utilized objects whose types were these classes: cin is an object of class istream and cout is an object of class ostream.

Therefore, we have as of now been utilizing classes that are identified with our record streams. Furthermore indeed, we can utilize our document streams the same way we are now used to utilize cin and cout, with the main distinction that we need to partner these streams with physical records.

Example:

```
#include <iostream>

#include <fstream>

using namespace std;

int main () {

   ofstream newFile;

   newFile.open (“sample.txt”);

   newFile << “Editing this file.\n”;
```

```
    newFile.close();

    return 0;

}
```

Writing To Document

This code makes a document called sample.txt and supplements a sentence into it in the same way we are utilized to do with cout, yet utilizing the record stream newFile.

Opening Files

The main operation for the most part performed on an object of one of these classes is to partner it to a genuine record. This system is known as to open a document. An open document is spoken to inside a system by a stream (an instantiation of one of these classes, in the past illustration this was myFile) and any operation performed on this stream object will be connected to the physical record related to it.

With a specific end goal to open a record with a stream object, we utilize its function open():

open (filename, mode);

Here, filename is the name of the file. Modes are given below for reference:

- ios::out - open for yield operations.
- ios::in - open for data operations.
- ios::ate - set the starting position at the end of the file.
- ios::binary - open in double mode.
- ios::trunc - In the event that the document opened for yield operations officially existed in the recent past, its past substance is erased and supplanted by the new one.
- ios::app - all yield operations are performed at the end of the document.

All these banners might be consolidated utilizing the bitwise administrator OR (|). Case in point, in the event that we need to open the record example.bin in binary mode to include information we could do it by the accompanying call to open():

Every one of the open() function of the classes ifstream, ofstream and fstream has a default mode that is utilized if the document is opened without a second operator. For ofstream and ifstream classes, ios::out and ios::in are naturally and individually accepted, regardless of the fact that a mode that does exclude them is passed as second parameter to the open() function.

The default value is just connected if the function is called without pointing out any quality for the mode parameter. On the off chance that the function is called with any value in that parameter the default mode is overridden, not joined.

Document streams opened in twofold mode perform output and input operations freely of any configuration parameters. Non-binary records are known as content documents, and a few interpretations may happen because of arranging of some exceptional characters (like newline and carriage return characters).

Since the first errand that is performed on a document stream article is for the most part to open a record, these three classes incorporate a constructor that consequently calls the open() function and has literally the same parameters as this part. Consequently, we could likewise have announced the past myfile protest and directed the same opening operation in our past case by composing:

ofstream myfile (“example.bin”, ios::out | ios::app | ios::binary);

Joining item development and stream opening in a solitary explanation. Both structures to open a document are legitimate and proportionate. To check if a record stream was fruitful opening a document, you can do it by calling to part is_open() with no contentions. This function gives back a bool estimation of valid in the case that without a doubt the stream

item is connected with an open record, or false overall:

Closing Files

When we are done with our operations on a record, we should close it so that its assets ended up accessible once more. So as to do that we need to call the stream's close() function. This function takes no parameters, and what it does is to flush the related handles and close the document:

newFile.close();

When this function is called, the stream object might be utilized to open an alternate record, and the document is accessible again to be opened by different methods. In the event that that an item is destructed while still connected with an open document, the destructor naturally calls close().

Text Files

Content document streams are those where we do exclude the ios::binary hail in their opening mode. These records are intended to store content and consequently all values that we enter or yield from/to them can endure some designing changes, which don't fundamentally compare to their strict binary value.

Notwithstanding eof(), which checks if the end of document has been arrived at, other functions exist to check the state of a stream (every one of them give back a bool value). In addition, there are several other functions like bad(), good() and eof() are available.

With a specific end goal to reset the state flags checked by any of these functions, we have recently seen we can utilize clear(), which takes no parameters.

The get and put Stream Pointers

All i/o streams have, no less than, one inward stream pointer. ifstream, in the same way as istream, has a pointer known as the get pointer that indicates the component to be perused

in the following information operation.

- Returns true if a perusing or composing operation fizzles. For instance, in the case that we attempt to keep in touch with a document that is not open for composing or if the gadget where we attempt to compose has no space cleared out.
- Returns valid in the same cases as bad (), additionally in the case that an organization mistake happens, in the same way as when an in sequential order character is concentrated when we are attempting to peruse a number.
- Returns true if a record open for perusing has arrived at the end.

It is the most nonexclusive state flag: it returns false in the same cases in which calling any of the past capacities would return genuine.

ofstream, in the same way as ostream, has a pointer known as the put pointer that indicates the area where the following component must be composed. At last, fstream, inherits both, the get and the put pointers, from iostream (which is itself determined from both istream and ostream). These inner stream pointers that indicate the perusing or composing areas inside a stream could be controlled utilizing the accompanying functions:

tellg() and tellp()

These two functions have no parameters and return a value of the pos_type, which is a whole number data type indicative of the current position of the get stream pointer (on account of tellg) or the put stream pointer (on account of tellp).

seekg() and seekp()

These functions permit us to change the position of the get and put stream pointers. Both are overloaded with two separate models. The principal model is:

seekg (position);

seekp (position);

Utilizing this model, the stream pointer is changed to indisputably the position (checking from the earliest starting point of the record). The sort for this parameter is the same as the one returned by capacities tellg and tellp: the part sort pos_type, which is whole number value.

The other model for these capacities is:

seekg (balance, heading);

seekp (balance, heading);

Utilizing this model, the position of the get or put pointer is situated to a balanced value with respect to some particular point controlled by the parameter course. Balance is of the data type off_type, which is additionally a number. Furthermore, bearing is of sort seekdir, which is a listed sort (enum) that decides the point from where balanced is tallied from, and that can take any of the accompanying qualities:

The accompanying case utilizes the part capacities we have recently seen to acquire the measure of a document:

- ios::beg - offset checked from the earliest starting point of the stream
- ios::cur offset - checked from the current position of the stream pointer
- ios::end - offset checked from the end of the stream

Binary Files

In binary records, to enter and output information with the extraction and insertion operators (<< and >>) and functions like getline is not effective, since we don’t have to organization any information, and information may not utilize the partition codes utilized by content documents to independent components. Record streams incorporate two functions particularly intended to output and input information successively: write and read.

Syntax:

write (memoryBlock, size);

read (memoryBlock, size);

The type of memoryBlock is “pointer to char” (char*), and indicates the location of a cluster of bytes where the read information components are put away or from where the information components to be composed are taken. The size parameter is a whole number esteem that points out the quantity of characters to be perused or composed from/to the memory square.

Example:

```
#include <fstream>
#include <iostream>
using namespace std;
char * memoryBlock;
ifstream::pos_type size;
int main () {
        ifstream newFile (“sample.bin”, ios::in|ios::binary|ios::ate);
        if (myFile.is_open())
        {
                mySize = newFile.tellg();
                memoryBlock = new char [size];
                newFile.seekg (0, ios::beg);
                newFile.read (memblock, size);
```

```
            newFile.close();
            cout << “the complete document substance is in memory”; delete[] memoryBlock;
        }
        else
            cout << “Not able to open document”;
        return 0;
}
```

In this illustration the whole document is perused and put away in a memory square.

Buffers and Synchronization

When we work with document streams, these are related to an inward support of type streambuf. This buffer is a memory block that is a intermediary between the stream and the physical record. The flushing of buffer compels it to write all the data available on it to the physical memory if the stream is output and free the memory if the stream is input. This process is called synchronization. It can be explicitly performed by calling the function sync() or with the help of manipulators.

Appendix

List of reserved keywords is as follows:

- auto
- asm
- break
- bool
- catch
- case
- class
- char
- const_cast
- const
- default
- continue
- do
- delete
- dynamic_cast
- double
- enum
- else
- export
- explicit
- false
- extern

- for
- float
- goto
- friend
- inline
- if
- long
- int
- namespace
- mutable
- operator
- new
- protected
- private
- register
- public
- return, short
- reinterpret_cast
- sizeof
- signed
- static_cast
- static
- switch
- struct
- this
- template

- true
- throw
- typedef
- try
- typename
- typeid
- unsigned
- union
- virtual
- using
- volatile
- void
- while
- wchar_t

In addition to the above-mentioned, there are some other operators that are also reserved under special conditions. These include:

- and_eq
- and
- bitor
- bitand
- not
- compl
- or
- not_eq
- xor

- or_eq
- xor_eq

Some compilers may also have some additional reserved keywords. You may read the compiler documentation for more information.

PHP Programming and MySQL For Beginners

A Simple Start To PHP & MySQL Written By A Software Engineer

Scott Sanderson

Table of Contents

Introduction

PHP is presumably the most commonly used scripting language in the website development world. It is most often utilized to upgrade pages. With PHP, you can do things like make user credentials for login pages like password and username. In addition, you can create check points of interest from a structure, create discussions, picture displays, overviews, and a ton more features. In the event that you've run over a website page that is based on PHP, then the writer has composed some programming code to liven up the plain, old HTML.

PHP is known as a server-side programming language. That is on account of the fact that PHP doesn't get executed on your machine. In fact, it gets executed on the machine you asked the page from. The results are then given over to you, and showed in your program. Other scripting languages that you must have heard about include ASP, Python and Perl.

The most common clarification of simply what PHP remains for is "Hypertext Pre-processor". However, that would make it HPP, without a doubt? An optional clarification is that the initials originate from the most punctual form of the system, which was called Personal Home Page Tools. This is exactly where the letters "PHP" have been taken from.

Anyhow, PHP is popular to the point that in case you're searching for a profession in the website development or web scripting industry, then you simply need to know this. In these book, we'll get you up and running. What's more, assuredly, it will be a much simpler than you anticipate.

Beginning With PHP

Before you can compose and test your PHP scripts, there’s one thing you’ll require - a server! Luckily, you don’t have to go out and purchase one. Actually, you won’t be using any additional cash. That is the reason PHP is so successfully used on the commercial platform. But, since PHP is a server-side scripting language, you either need to get some web space with an organization that supports PHP, or make your machine imagine that it has a server. This is on the grounds of the fact that PHP is not running on your PC - it’s executed on the server. The results are then sent once more to the customer PC (your machine).

Don’t stress if this sounds a bit of overwhelming - we’ve run over a less demanding approach to get you up and running. We’re going to be utilizing some product called Wampserver. This permits you to test your PHP scripts on isolated systems. It introduces all that you require, on the off chance that you have a Windows PC. We’ll clarify how to get it introduced in a minute, and where to get it from. However, here is some quick advice for users of non-windows systems.

Apple Users

If you are an apple user, you can try the following links for setting up your system for running PHP.

http://www.onlamp.com/pub/a/mac/2001/12/07/apache.html

in your attempts to get your system running smoothly, keep a stern check on the location of file storage and the address of localhost.

Linux Users

Secondly, if you work on a Linux system, then setting up your system for PHP can be tricky because there are a very few online resources available for your help. You can try your luck with the below mentioned links for some help.

http://www.php-mysql-tutorial.com/wikis/php-tutorial/installing-php-and-mysql.aspx

Windows Users

Doing PHP programming is simpler for you if you are a Windows user, considering the fact that plenty of online help is just a click away. The first step for setting up your Wampserver is to download the software. You can search for it on Google and download it using the official website.

Assuredly, you have now downloaded Wampserver and are well acquainted with the software. This will provide for you a server at an isolated PC (Windows clients). So, now you are equipped enough to test your scripts. Assuming that you have the Wampserver setup with you, you can install it by following the instructions.

Once the main panel appears, look around the menu items for tabs for stopping and starting the server. Other links shall take you to document viewer and system help, in addition to other functionalities associated with the system. Click on localhost, and you'll see a page. Localhost simply alludes to the server running on your standalone machine. An alternate approach to allude to your server is by utilizing the IP address 127.0.0.1.

Your server should be up and running smoothly by now. If you are facing any issues, you can look for their solutions on online forums or seek expert help.

Variables in PHP

A variable is simply a stockpiling area. You place things into your stockpiling zones (variables) so you can utilize and control them within your projects. Things you generally need to store are numbers and content.

In case you're alright with the thought of variables, then you can proceed onward. If not, consider the following explanation. Assume you need to create a record of your attires. You hire two individuals to help you, a man and a lady. These two individuals are going to be your stockpiling zones. They are going to hold things for you, while you count up what you possess. The man and the lady, then, are variables.

You check what number of coats you have, and afterward offer these to the man. You tally what number of shoes you have, and offer these to the lady. Unfortunately, we have a terrible memory. The inquiry is, which one of your kin (variables) holds the layers and which one holds the shoes? To help you recollect that, you can give your kin names! You could call them something like this:

miss_shoes

mr_coats

Be that as it may, it's altogether up to you what names you give your kin (variables). On the off chance that you like, they could be called this:

boy_coats

girl_shoes

But since your memory is terrible, it's best to provide for them names that help you recollect what it is they are holding for you. There are a few things your kin shrug off being called. You can't start their names with an underscore (_), or a number. Anyhow

most different characters are fine.

Alright, so your kins (variables) now have names. Be that as it may it’s horrible simply providing for them a name. They are going to be doing some work for you, so you have to let them know what they will be doing. The man is going to be holding the coats. Anyhow we can detail what number of covers he will be holding. On the off chance that you have ten layers to provide for him, then you do the “telling” him this using the syntax shown below:

mr_coats = 10

In this way, the variable name starts things out, followed by an equivalent sign. After the equivalent sign, you tell your variable what it will be doing. Holding the number 10, for our situation. The equivalents sign, incidentally, is not so much an equivalent sign. It’s called a task administrator. Anyway, don’t stress over it, at this stage. Simply recall that you require the equivalents sign to store things in your variables.

Notwithstanding, you’re learning PHP, so there’s something missing. Two things, really. Initially, your kin (variables) require a dollar sign toward the starting (individuals are similar to that). So the actual statement should actually be similar to the statement given below.

$mr_coats = 10

In the event that you miss the dollar sign out, then your kin will decline to work! Yet the other thing missing is something truly particular and fastidious - a semi-colon. Lines of code in PHP need a semicolon toward the end, as shown in the statement below.

$mr_coats = 10;

In the event that you get any parse slips when you attempt to run your code, the first thing to check is whether you’ve missed the semicolon off the end. It simple to do, and can be

disappointing. The following thing to check is whether you've passed up a major opportunity of finding a dollar sign. So the man is holding ten covers. We can do likewise thing with the other individual (variable) as well.

$miss_shoes = 20;

In this way, $miss_shoes is holding an estimation of 20. On the off chance that we then needed to include what number of things or items we have as such, we could set up another variable (Note the dollar sign at the beginning of the new variable):

$total_items

We can then include the covers and the shoes. So as to add in PHP, you can utilize a statement like the one shown below:

$total_items = $mr_coats + $miss_shoes;

Keep in mind, $mr_coats is holding an estimation of 10, and $miss_shoes is holding an estimation of 20. On the off chance that you utilize a plus sign for addition, PHP supposes that you need to add. So, it will work out the aggregate for you. The answer will then get put away in our new variable, the one we've called $total_items. A syntax equivalent of this statement is given below.

$total_items = 10 + 20;

Once more, PHP will see the addition sign and add the two together for you. Obviously, you can include more than two things to this addition formula.

$total_items = 10 + 20 + 9 + 50 + 1100;

Be that as it may, the thought is the same. PHP will see the addition signs and afterward include things up. The answer is then put away in your variable name, the one to the left of the equal sign.

Placing Text into PHP Variables

In the past segment, you perceived how to place numbers into variables. Be that as it may, you can likewise place content into your variables. Assume you need to know something about the coats or layering clothes that you possess. You can categorize our coats into categories like Winter layers, Coats or Summer layers. You can choose to index them, also. You can put immediate content into your variables. It a comparative approach to putting away numbers:

$coat_1 = ” Summer Coats”;

Once more, our variable name begins with a dollar sign ($). We’ve then provided for it the name coat_1. The ‘equal to’ sign takes after the variable name. After the sign, then again, we have content - Winter Coats. Anyhow, recognize the twofold quotes around our content. On the off chance that you don’t encompass your content within quotes, you’ll get unpredictable results. You can, in any case, use single quotes rather than twofold quotes, as shown in the statement below.

$coat_1 = Summer Coats’;

Be that as it may you can’t do this:

$coat_1 = Summer Coats”;

In the above line, we’ve begun with a solitary quote and finished with a twofold quote. This will get you a mistake.

We can store other content in the same way:

$coat_2 = “Coats”;

$coat_3 = “Winter Coats”;

The content will then get put away in the variable to the left of the ‘equal to’ sign.

In this way, to recap, variables are zones, which possess a well-defined capacity. You utilize these capacity territories to control things like content and numbers. You’ll be

utilizing variables a great deal, and on the following few pages you'll perceive how they function in practice.

Using Variables With PHP

To begin with, we'll examine how to access what's in your variables. We're going to be reviewing our results on a site page. So check whether you can get this script working in the first place, in light of the fact that it's the one we'll be expanding on later in this chapter. Utilizing a content manager like your PHP programming or Notepad, can be helpful. You can duplicate and glue it, on the off chance that you lean toward. Anyway you take in more by writing it out yourself - it doesn't generally soak into your mind unless you're committing errors!

```
<html>
<head>
<title>Practicing Variables</title>
</head>
<body>
<?php print("It is working!"); ?>
</body>
</html>
```

When you are done with writing everything, finish by saving the page as variable.php. At that point, simply run the script. Keep in mind: when you're saving your work, place it in the WWW organizer. To run the page, begin your program up and sort this in the location bar:

http://localhost/variable.php

On the off chance that you've made an organizer inside the www envelope, then the

location to sort in your program would be something like:

http://localhost/Foldername/variable.php

Besides this, you ought to have seen the content “It is working!” showing up on as a result of your program. Assuming this is the case, Congratulations! You have a working server, which is up and running! In case you’re utilizing Wampserver, you ought to see a symbol in the lowest part right of your screen. Click the symbol and select Start All Services from the menu of the task bar.

Out of the complete program, the line of importance is:

<?php print(“It is working!”); ?>

Whatever is left of the script is out and out HTML code. How about we analyze the PHP in more detail. We’ve put the PHP in the BODY segment of a HTML page. Scripts can likewise, and frequently do, go in the HEAD area of a HTML page. You can likewise compose your script without any HTML. Anyway, before a program can perceive your script, it needs some assistance. You need to let it know what sort of script it is. Programs perceive PHP by searching for this accentuation (called syntax):

<?php ?>

So you require a left edge section (<) then an inquiry mark (?). After the inquiry imprint, write PHP (in upper or lowercase). After your script has completed, put an alternate inquiry mark. At last, you require a right plot section (>). You can put to the extent that as you like between the opening and shutting sentence structure.

To show things on the page, we’ve utilized print( ). What you need the program to print goes between the round brackets. In case you’re printing string content, then you require quotes (single or twofold quotes). To print what’s within a variable, simply prefix the variable name with a dollar sign. At last, the line of code finishes as ordinary - with a

semicolon (;).

Presently how about we adjust the essential page with the goal that we can set up a few variables. We'll attempt some content first. Keep the HTML as it may be, yet change your PHP from Code 1 to Code 2.

Code 1:

<?php print("It is working!"); ?>

Code 2:

<?php

print("it Worked!");

?>

Alright, it's a useless change! Anyhow, spreading your code out over more than one line makes it simpler to see and comprehend what you're doing. The present code only has a single line. In the following code, we have added some more code to the sample script.

<?php

$testing_string = "It is working!";

print("It is working!");

We've set up a variable called $testing_string. After the equivalents sign, the content "It is working!" has been included. The line is then finished with a semicolon. However, before running the script, you must add the following print line to the code:

print($testing_string);

At that point include a few comments

<?php

/ -Variables in PHP Practice-

$testing_string = “It is working!”;

print($testing_string);

?>

Comments in PHP are for your profit. They help you recall what the code should do. A comment can be included by writing two slices. This advises PHP to disregard whatever remains of the line. After the two slices, you can write anything you like. An alternate approach to include a comment makes use of the following format:

/* <comment

*/

Utilize this kind of commenting on the off chance that you need to overflow to more than one line. Whichever technique you pick, verify you add comments to your code: they truly do help. This is true particularly on the off chance that you need to send your code to another person!

How could you have been able to you get on? You ought to have seen that precisely the same content got printed to the page. Furthermore you may be supposing - what’s the major ordeal? All things considered, what you simply did was to pass some content to a variable, and afterward have PHP print the data of the variable. It’s an enormous step: your coding profession has now started!

You can join together coordinate content, and whatever is in your variable. The full stop (period or dab, to a few) is utilized for this. Assume you need to print out the following:

“My variable contains the estimation of 5”. In PHP, you can do it using the code given below:

<?php

```
$number_1 = 5;

$value_text = ‘My variable contains the estimation of ‘;

print($value_text . $number_1);

?>
```

So now we have two variables. The new variable holds our string content. When we’re printing the value of both variables, a full stop is utilized to distinguish the two. Go for the above script, and see what happens. Presently erase the dab and afterward attempt the code once more. Any lapses?

You can likewise do this kind of thing:

```
<?php

$number_1 = 5;

print (‘My variable contains the estimation of ” . $number_1);

?>
```

This time, the immediate content is not inside a variable, yet recently included in the print explanation. Again a full stop is utilized to distinguish the string content from the variable name. What you’ve recently done is called linking or connecting. Attempt the new script and see what happens.

Addition

Alright, how about we do some adding with variables. To add in PHP, the addition operator or ‘+’ sign is utilized. On the off chance that you have the code open from the previous section, take a stab at changing the full stop to an addition sign (+). Run the code, and see what happens.

To add the values of the variables, you simply separate every variable name with an in

addition operator. Attempt this new script:

```
<?php

$number_1 = 4;

$number_2 = 3;

$sum_value = $number_1 + $number_2;

$value_text = ‘Result = ‘;

print ($value_text . $sum_value);

?>
```

In the above script, we’ve included a second number, and doled out a value to it.

```
$number_2 = 3;
```

A third variable is then announced, which we’ve called $sum_value. To the right of the equivalents sign, we’ve included the value of the first variable and the value of the second variable:

```
$sum_value = $number_1 + $number_2;
```

PHP comprehends what is within the variables called $number_1 and $number_2, in light of the fact that we’ve quite recently let it know in the two lines above! It sees the addition operator, and then adds the two variable values together. It puts the response of the addition in the variable to the left of the equivalents sign (=), the one we’ve called $sum_value.

To print out the answer, we’ve utilized linking:

```
print ($value_text . $sum_value);
```

This script is somewhat more entangled than the ones you’ve been doing. In case you’re a bit astounded, simply recollect what it is we’re doing: adding the value of one variable to

the same of an alternate. The critical line is this one:

$sum_value = $number_1 + $number_2;

The addition operation to the right of the equivalents sign gets computed first ($number_1 + $number_2). The aggregate of the operation is then put away in the variable to the left of the equivalents sign ($sum_value). You can, obviously, include more than two numbers.

Subtraction

We’re not going to weigh things around subjecting you to torrents of substantial Math! In any case you do need to know how to utilize the fundamental operators. First and foremost up is subtracting. Subtraction is pretty much the same as addition, which you did in the previous section. Rather than the addition sign (+), you need to utilize the minus sign (-). Change your $sum_value line to this, and run your code:

$sum_value = $number_1 - $number_2;

The $sum_value line is pretty much the same as the first. But we’re currently utilizing the minus sign instead of the addition operator. When you run the script you ought to, obviously, get the answer 1. Once more, PHP recognizes what is within the variables called $number_1 and $number_2. It knows this in light of the fact that you relegated data values to these variables in the initial two lines.

At the point when PHP runs over the minus sign, it does the subtraction for you, and puts the answer into the variable on the left of the equivalents sign. We then utilize a print function to show what is within the variable. Much the same as addition, you can subtract more than one number at once. Attempt this:

<?php

$number_1 = 3;

$number_2 = 4;

```
$number_3 = 30;

$sum_value = $number_3 - $number_2 - $number_1;

print ($sum_value);

?>
```

The answer you ought to get is 23. You can likewise blend addition operations with subtraction. Here’s an illustration:

```
<?php

$number_1 = 3;

$number_2 = 4;

$number_3 = 30;

$sum_value = $number_3 - $number_2 + $number_1;

print ($sum_value);

?>
```

Run the code above. What answer did you get? Is it true that it was the answer you were anticipating? Why do you think it printed the number it did? On the off chance that you thought it may have printed an alternate response to the one you got, the reason may be the way we set out the whole expression. Did we mean 30 - 4, and after that add 3? Alternately, did we mean add 3 and 4, and then subtract it from 30? The principal entirety would get 29, yet the second total would get 23.

To illuminate what you mean, you can utilize enclosures as a part of your wholes and parts of expressions. Here’s the two separate forms of the expression. Attempt them both in your code. Be that as it may, note where the enclosures are:

Form one

$sum_value = ($number_3 - $number_2) + $number_1;

Form two

$sum_value = $number_3 - ($number_2 + $number_1);

It’s generally a decent thought to utilize brackets within your holes, just to clear up what you need PHP to compute. That way, you won’t get an exceptional answer!

An alternate motivation to utilize brackets is a direct result of something many refer to as operator precedence. In PHP, a few operators (particularly mathematical functions) are figured before others. This implies that you’ll get answers that are totally surprising! Therefore, be sure to verify the order of execution of the operator used in an expression before anticipating any correct results.

Multiply

To multiply in PHP and pretty much every other programming language, the * operator is utilized. On the off chance that you see 3 * 4, it means duplicate 3 by 4. Here’s some code for you to attempt:

<?php

$number_1 = 10;

$number_2 = 20;

$sum_value = $number_2 * $number_1;

print ($sum_value);

?>

In the above code, we’re simply multiplying whatever is within our two variables. We’re then allotting the response to the variable on the left of the equivalents sign. You can most likely think about what the answer is without running the code! Much the same as

subtraction and addition, you can multiply more than two numbers

Division

To divide one number by an alternate, the / operator is utilized within PHP. On the off chance that you see 25/5, it means isolate 5 into 25. Attempt it yourself using the code given below:

```
<?php

$number_1 = 5;

$number_2 = 25;

$sum_value = $number_2/$number_1;

print ($sum_value);

?>
```

Once more, you must be watchful of operator precedence. Attempt this code:

```
<?php

$number_1 = 5;

$number_2 = 25;

$number_3 = 10;

$sum_value = $number_3 - $number_2/$number_1;

print ($sum_value);

?>
```

PHP won’t work out the entirety from left to right! Division is carried out before subtraction. So this will accomplish first:

$number_2/ $number_1

Also NOT this:

$number_3 - $number_2

Utilizing brackets will clear things up things and you will be able to perform the operations you want and get the correct results.

Working With Floating Point Numbers

A floating point number is one that has a speck in it, in the same way as 0.5 and 10.8. You needn’t bother with any unique sentence structure to set these sorts of numbers up. Here’s a case for you to attempt:

<?php

$number_1 = 3.6;

$number_2 = 1.4;

$sum_value = $number_2 + $number_1;

print ($sum_value);

?>

You can multiplication, subtraction, addition and division of numbers in precisely the same path as the whole numbers you’ve been utilizing. A cautioning accompanies the use of floating point numbers. You shouldn’t believe them, in case you’re after a ridiculously exact answer!

You saw in the last area that variables are capacity regions for your content and numbers. At the same time the reason you are putting away this data is with the goal that you can do something with them. On the off chance that you have put away a username in a variable, for instance, you’ll then need to check if this is a legitimate username. To help you do the checking, something many refer to as Conditional Logic comes in as something

exceptionally helpful. In the following segment, we’ll investigate simply what Conditional Logic is and how it works.

Conditional Logic

Conditional Logic is about inquiring as to whether a condition is true or false and on the basis of the result of the condition evaluation, the corresponding code is executed. When you press a button named “Don’t Press this Button – at any cost!” you are utilizing Conditional Logic. You are asking, “Admirably, what happens IF I do click the button?”

You utilize Conditional Logic as a part of your everyday life constantly:

“On the off chance that I turn the volume up on my stereo, will the neighbours be angry?”

“On the off chance that use all my cash on another pair of shoes, will it make me cheerful?”

“In the event that I study this course, will it enhance my site?”

Conditional Logic uses the “IF” word a considerable measure of times. Generally, you utilize Conditional Logic to test what is contained in a variable. You can then settle on choices focused around what is within the variable. As an illustration, contemplate the username once more. You may have a variable like this:

$username = “regular_visitor”;

The content “regular_visitor” will then be put away within the variable called $username. You would utilize some Conditional Logic to test whether the variable $username truly does contain the name of one of your general guests. You need to ask:

“If $username is bona fide, then let $username have entry to the site.”

In PHP, you utilize the “IF” word like this:

```
if ($username == “allowed”) {

//Code to let client get to the site here;
```

}

Without any checking, the if articulation follows the following syntax:

if (<condition to be checked>) {

//Code

}

You can see it all the more plainly, here. To test a variable or condition, you begin with the saying “if”. You then have a couple of round sections. You additionally require some more sections - wavy ones. You require the left wavy section first { and after that the right wavy section } toward the end of your if construct. Get them the wrong path round, and PHP declines to work. This will get you a lapse:

if($username == “allowed”) }

//Code to Let client get to the site here

{

Along these lines, the following code can also be tested:

if($username == “allowed”) {

/Code to Let client get to the site here;

{

The first has the wavy sections the wrong route round (ought to be left then right), while the second one has two left wavy sections.

In the middle of the two round sections, you write the condition you need to test. In the illustration above, we’re trying to see whether the variable called $username has an value, “allowed”:

($username = “allowed”)

Once more, you'll get a lapse in the event that you don't get your round sections right! So the language structure for the if proclamation is this:

```
if(testing_condition) {

//Code to be executed if the condition is true

}
```

In the event that you have to test two conditions identified with the same variable, if-else build can be utilized. The linguistic structure for the if else construct is this:

```
if (testing_condition) {

//code if condition is true

}

else {

//Code if condition is false

}
```

In the event that you take a gander at it closely, you'll see that you have an ordinary if Statement initially, which is followed by an "else" part. Here's the "else" part:

```
else {

//Code to be executed if the condition is false

}
```

Once more, the left and right wavy sections are utilized. You can likewise include "else if" parts to the if statements you've been investigating in the past segments. The syntax to be utilized for the purpose is given below:

```
else if (testing_condition) {
```

//Corresponding Code

}

Relational Operators

You saw in the last segment how to test what is within a variable. You utilized if, else … if, and else. You also made use of the twofold equivalents sign (==) to test whether the variable was the same in value as the given condition. The twofold equivalents sign is known as a Comparison Operator. There a many of these “operands” that you can use for testing. Here’s a rundown of the list of operators and what they can be used for. Investigate, and after that we’ll see a couple of illustrations of how to utilize them.

Operator	**Purpose**	**Syntax**
==	Equal	a == b
!=	Not equal	a != b
>	Greater than	a > b
<	Lesser than	a < b
>=	Greater than or equal to	a >= b
<=	Lesser than or equal to	a <= b

Here’s some more data on the above Operands.

== (Has the same value as)

The twofold equivalents sign can signify “Has an estimation of” or “Has the same value as”. In the sample underneath, the variable called $variable_1 is continuously contrasted with the variable called $variable_2

if ($variable_1 == $variable_2) {

}

!= (Is NOT the same value as)

You can likewise test if one condition is NOT the same as an alternate. In this case, you require the shout imprint/equivalents sign mix (!=). In the event that you were trying for an authentic username, for instance, you could say:

if ($which_user_entered != $user_name) {

print(“you’re not a authenticated client of this site!”);

}

The above code says, “If what the client entered is NOT the same as the quality in the variable got, $user_name, then print something out.

< *(Less Than)*

You’ll need to test if one worth is short of what an alternate is. Utilize the left edge section for this (<)

> *(Greater Than)*

You’ll additionally need to test if one worth is more than an alternate. Utilize the right point section for this (>)

<= *(Less than or equivalents to)*

For somewhat more accuracy, you can test to check whether one variable is short of or equivalent to an alternate. Utilize the left point section took after by the equivalents sign (<=)

>= *(Greater than or equivalents to)*

In the event that you have to test if one variable is more than or equivalent to an alternate, utilize the right point section took after by the equivalents sign (>=)

Switch – Case

For a real world conditional logic system, a long rundown of if and else … if proclamations will be utilized. A superior alternative to this structure when testing a single variable is the Switch-Case construct. To perceive how switch case function, think about the accompanying code:

```
<?php

$new_picture =tower;

switch ($new_picture) {

case ‘dog’:

print(Dog Picture’);

break;

case ‘tower’:

print(Tower Picture’);

break;

}

?>
```

In the code above, we put the string content “tower” into the variable called $new_picture. It’s this content that we need to check. We need to comprehend what is within the variable, so we can show the right picture.

To test a solitary variable with a Switch Statement, the accompanying language structure is utilized:

```
switch ($name_of_variable) {

case ‘testing_value’:
```

//code

break;

}

It looks a bit complex, so we'll separate it.

switch ($name_of_variable) {

You Start with the expression "switch" then a couple of round sections. Within the round brackets, you write the name of the variable you need to check. After the round sections, you require a left wavy bracket.

case 'testing_value':

The expression "case" is utilized before each one quality you need to check for. In our code, a rundown of qualities was originating from a drop-down menu. These quality were: tower and dog, among others. These are the qualities we require after the statement 'case'. After the content or variable you need to check for, a colon is required (:).

//code here

After the colon on the "case" line, you write the code you need to execute. Evidently, you'll get a mistake in the event that you pass up the opportunity of adding any semicolons toward the end of your lines of code!

break;

You have to advise PHP to "Break out" of the switch explanation. In the event that you don't, PHP will basically drop down to the following case and execute every case that follows the matched case. Utilize the expression "break" to escape from the switch articulation.

In the event that you take a gander at the last few lines of the Switch Statement in this

document, you'll see something else also that you can add to your code:

default:

print ("No Image Selected");

The default alternative is similar to the else from if … else. It's utilized when there could be other, obscure, alternatives. A kind of "get all" alternative.

Logical Operators

And also the PHP correlation administrators you saw prior, there's likewise something many refer to as Logical Operators. You commonly utilize these when you need to test more than one condition at once. Case in point, you could verify whether the username and secret key are right from the same If Statement. Here's the table of these Operands.

Operator	**Purpose**	**Usage**
&&	Both condition should be true	a && b
||	One or both of the conditions should be true	a || b
AND	Both conditions are true	a AND b
XOR	Only one of the conditions is true	a XOR b
OR	One or both conditions are true	a OR b
!	Negation	!a

The new Operands are fairly unusual, in case you're reaching them with confusion. A few of them even do likewise thing! They are extremely helpful, however, so here's a more intensive look.

The && Operator

The && operator mean AND. Utilize this on the off chance that you require both conditions to be valid, as in our username and password testing. Truth be told, you would

prefer not to give individuals access on the off chance that they simply get the username right or just the right password. Here's a sample code to illustrate the concept:

$user_name = 'registered_user';

$pass_word ='secret_password';

if ($user_name =='registered_user' && $pass_word =='secret_password') {

print("Welcome User!");

}

else {

print("Invalid username/password!");

}

The if explanation is used in the same manner as before. However, perceive that now two conditions are continuously tried:

$user_name == 'registered_user' && $pass_word ='secret_password'

This says, "If username is right AND the secret key is right, as well, then give the user access." Both conditions need to go between the round sections of your if proclamation.

The | Operator

The two straight lines mean OR. Utilize this operator when you just need one of your conditions to be true. For instance, assume you need to allow a markdown to individuals in the event that they have used more than 500 pounds OR they have an extraordinary key. Else they don't get any rebate. You'd then code like this:

$total_expenditure =500;

$special_coupon_key ='98765';

```
if ($total_spent == 500 | $special_coupon_key == ‘98765’) {

print(“Discount applies!”);

}

else {

print(“No discount!”);

}
```

This time we’re trying two conditions and just need ONE of them to return a true value. In the event that both of them are true, the code gets executed. On the off chance that they are both false, then PHP will proceed onward.

AND and OR

These are the same as && and || operators, discussed in the previous sections. However, there is an unobtrusive contrast. As a novice, you can just use the following format for implementing AND:

```
$user_name == ‘registered_user’ && $pass_word ==‘secret_password’
```

The following statement also means the same as:

```
$user_name == ‘registered_user’ AND $pass_word ==‘secret_password’
```

Which one you use is dependent upon your preference. AND is a ton simpler to peruse than &&. Similarly, OR is a great deal less demanding to peruse than ||. The distinction, by the way, is to do with Operator Precedence. We touched on this when we examined variables, prior. It shall be discussed in detail in the sections to follow

XOR

You likely won’t require this one excessively. Anyway, its utilized when you need to test if

one estimation of two is genuine however NOT both. On the off chance that both conditions are the same, then PHP sees the declaration as false. On the off chance that they are both diverse, then the answer is true. Assume you needed to pick a victor between two hopefuls. Stand out of them can win. It’s a XOR circumstance!

```
$contestant_1 = true;

$contestant_2 = true;

if($contestant_1 XOR $contestant_2) {

print(“There can be only one winner!”);

}

else {

print(“Both cannot be winners!”);

}
```

Check whether you can figure which of the two will print out, before running the script.

The ! Operator

This is known as the NOT operator. You utilize it to test whether something is NOT something else. You can likewise utilize it to turn around or negate the value of a true or false value. For instance, if you need to reset a variable to true, on the off chance that it’s been set to false, and the other way around, you can use this operator for best results. Here’s some code to attempt:

```
$not_value = false;

if($not_value == false) {

print(!$not_value);

}
```

The code above will print out the number 1. You'll see why when we handle Boolean values underneath. What we're stating here is, "If $test_value is false then the code must print what its NOT." What its NOT is true, so it will now get this quality. A bit confounded? It's a dubious one. However, it can prove to be useful!

Boolean Values

A Boolean value is one that can hold one of the following states: true and false. True is typically given an estimation of 1, and False is given an estimation of zero. You set them up much the same as different variables:

$t_value = 1;

$f_value = 0;

You can supplant the 1 and 0 with the words "true" and "false" (without the quotes). Yet a note of alert, in the event that you do. Attempt this script out, and see what happens:

<?php

$f_value = false;

$t_value = true;

print (" f_value = " . $f_value);

print ("t_value = " . $t_value);

?>

What you ought to discover is that the t_value will print "1", however the f_value won't print anything. Presently, supplant true with 1 and false with 0, in the script above, and see what prints out. Boolean values are extremely normal in programming, and you regularly see this kind of coding:

$t_value = true;

```
if ($true_value) {

print(“True Value”);

}
```

This is a shorthand method for saying “if $t_value holds a Boolean estimation of 1 then the print statement must be executed”. This is the same as:

```
if ($t_value == 1) {

print(“True”);

}
```

The NOT operand is additionally utilized a great deal with this sort of if articulation:

```
$t_value = true;

if (!$t_value) {

print(“True”);

}

else {

print(“False”);

}
```

You’ll likely meet Boolean values a great deal, amid your programming life. It’s better to get a hang of them!

Operator Precedence

Here’s a rundown of the operators you’ve met as such, and the request of priority. This can have any kind of effect, as we saw amid the numerical computations. Don’t stress over these excessively, unless you’re persuaded that your math or coherent is right. In which

case, you may need to counsel the accompanying:

The main operators you haven’t yet met on the rundown above are the = and != operators.

In late releases of PHP, two new operators have been presented: the triple equivalents sign (=) and a twofold equivalent (!=). These are utilized to test if one variable has the same value as an alternate AND the two are of the same type. A case would be:

$number_val = 2;

$text_val = ‘two’;

if ($number_val === $text_val) {

print(“Both are same!”);

}

else {

print(“They are different!”);

}

So this asks, “Do the variables match precisely?” Since one is content and the other is a number, the answer is “no”, or false. We won’t be utilizing these administrators much, but it always beneficial to know the options you have with you.

The operator precedence is as follows (highest priority to lowest priority):

- / % *
- - + .
- > >= < <=
- !== ===
- &&
- II

- AND
- OR
- XOR
- OR

PHP AND HTML FORMS

On the off chance that you know a little HTML, you will realize that the FORM labels can be utilized to interact with your clients. Things that can be added to a structure in the form of radio buttons, text boxes, drop down menus, checkboxes and submit buttons. A fundamental HTML structure with a text box and a submit button can be created in the following manner:

```
<html>

<head>

<title>a BASIC Form in HTML</title>

</head>

<body>

<form NAME =“form_1” METHOD =” ” ACTION = ””>

<input TYPE = “text” VALUE =“user_credentials”>

<input TYPE = “Submit” Name = “Submit_1” VALUE = “Login_value”>

</Form>

</body>

</html>
```

We won’t clarify what all the HTML components do, as this is a book on PHP. Some commonality with the above is expected. Yet, we’ll examine the METHOD, ACTION and SUBMIT properties in the structure above, in light of the fact that they are imperative.

In the event that a client goes to your site and needs to login, for instance, then you’ll have to get the subtle elements from textboxes. When you have the text that the client entered,

you then test it against a list of your clients. This list of clients is normally put away on a database, which we'll perceive how to code for in a later section. To start with, you have to think about the HTML properties METHOD, ACTION and SUBMIT.

METHOD Attribute

On the off chance that you take a gander at the first line of our structure from the last section, you'll perceive the use of an attribute called METHOD:

<form NAME ="form_1" METHOD =" " ACTION = "">

The Method ascribe is utilized to tell the program how the data ought to be sent. The two most well known techniques you can utilize are GET and POST. In any case our METHOD is clear. So transform it to this:

<form NAME ="form_1" METHOD ="get" ACTION = "">

To see what impact utilizing GET has, save your work again and after that click Submit on your form. The thing to recognize here is the location bar. The address would have been appended by the following string:

?Submit_1=login_value

This is an outcome of utilizing the GET system. The information from the structure winds up in the location bar. You'll see an inquiry imprint, emulated by information. Submit_1 was the NAME of the button on the form, and Login_value was the VALUE of the button. This is what is returned by the GET technique. You utilize the GET system when the information you need returned is not pivotal data that needs securing. You can likewise utilize POST as the Method, rather than GET. Read underneath to see the distinction.

POST Attribute

In the past section, you saw what happened in the program's location bar when you utilized the GET technique for Form information. The option to GET is to utilize POST.

Change the first line of your FORM to this:

<form NAME =“form_1” METHOD =“post” ACTION = ””>

Shut your program down, and open it again. Load your page once more, and after that click the button. Your location bar will then resemble this not have the ?Submit1=login part anymore. That is on account of how we utilized POST as the strategy. Utilizing POST implies that the structure information won’t get affixed to the location in the location bar for all to see.

We’ll utilize both POST and GET all through the book. Be that as it may, it relies on upon the undertaking. If the information is not delicate then utilize GET, overall utilization POST. An alternate paramount quality of the Form tag is Action. Without Action, your structures won’t go anyplace! We’ll perceive how this works in the following part.

ACTION Attribute

The Action property is an essential attribute of your form. That is to say, “Where do you need the structure sent?”. On the off chance that you miss it out, your structure won’t go anyplace. You can send the structure information to an alternate PHP script, the same PHP script, an email address, a CGI script, or some other manifestation of script.

In PHP, a prominent strategy is to send the script to the same page that the structure is on – send it to itself, as such. So you have to change the structure of the form that you have been using until now. Place the accompanying, and change the ACTION line to this:

<form Name =“form_1” Method =“post” ACTION = “next_page.php”>

So we’re going to be sending the structure information to precisely the same page as the one we have stacked – to itself. We’ll put some PHP on the page to handle the structure information. Anyway for the present, save your work again and after that click on submit. You won’t see anything diverse, yet you shouldn’t see any slip message either! When your

script has an Action attribute set, you can then Submit it. Which we'll see in the following part.

SUBMIT Button

The HTML Submit is utilized to submit information to the script specified in ACTION. Sample code to illustrate how this works is given below:

<form Name ="form_1" Method ="post" ACTION = "next_page.php">

So the page specified in the ACTION quality is next_page.php. To Submit this script, you require a HTML Submit button:

<input TYPE = "Submit" Name = "Submit_1" VALUE = "Login_value">

You don't have to do anything uncommon with a Submit. All the submitting is carried out. If the length of SUBMIT has an ACTION set, then your information will be sent to some place. In any case the NAME characteristic of the Submit comes in as something exceptionally helpful. You can utilize this Name to test if the structure was truly submitted, or if the client recently clicked the refresh button.

This is critical when the PHP script is in agreement with the HTML structure. Our Submit is called "Submit_1", yet you can call it just about anything you like. Since you know all about METHOD, ACTION, and SUBMIT now, we can proceed onward to preparing the information that the client entered. To begin with, we shall explore how to get text entered by the user in the next section.

Text Boxes

You already know about METHOD and ACTIOn attributes by now. The METHOD characteristic lets you know how information is constantly sent, and the ACTION trait lets you know where it is to be sent. To get at the content that a client entered into a text area, the text box needs a NAME trait. You then tell PHP the NAME of the textbox you need to

work with. Our text box hasn't got a NAME yet, so change your HTML to this:

<input TYPE = "text" VALUE ="username" NAME = "user_name">

The NAME of our textbox is user_name. It's this name that we will be utilizing as a part of a PHP script. To return information from a HTML structure, you utilize the accompanying statement:

$_post['element_name'];

You can appoint this to a variable:

$variable_name = $_post['element_name'];

Before we clarify all the language structure, add the accompanying PHP script to the HTML code you have. Make a point to include it the HEAD area of your HTML:

<html>

<head>

<title> My First HTML Form</title>

<?php

$my_user_name = $_post['user_name'];

print ($my_user_name);

?>

</head>

Save your work once more, and click the submit button to run your script. You ought to see this show up over your text box:

Erase the text "user_name" from the textbox, and click the button once more. Your new text ought to show up over the textbox. The content box itself, nonetheless, will in any

case have “user_name” in it. This is on the grounds that the text box is getting reset when the information is come back to the program. The Value of the text box is what is continuously shown.

So how can it function?

The $_post[ ] is an inbuilt capacity you can use to get POST information from a structure. In the event that you had METHOD = “GET” on your structure, then you’d utilized this:

$my_user_name = $_get[‘user_name’];

So you start with a dollar sign ($) and an underscore character (_). Next comes the METHOD you need to utilize, POST or GET. You have to sort a couple of square sections next. In the middle of the square sections, you write the NAME of your HTML structure component – username, for our situation.

$_post[‘user_name’];

Obviously, you require the semi-colon to finish the line. Whatever the VALUE was for your HTML component is the thing that gets returned. You can then dole out this to a variable:

$my_user_name = $_post[‘user_name’];

So PHP will search for a HTML structure component with the NAME user_name. It then takes a gander at the VALUE characteristic for this structure component. It gives back this worth for you to utilize and control. Right now, everything we’re doing is returning what the client entered and printing it to the page. Anyway, we can utilize a bit of Conditional Logic to test what is within the variable. As an illustration, change your PHP to this:

$my_user_name = $_post[‘user_name’];

if ($my_user_name == “letme”) {

print (“Welcome User!”);

}

else {

print (“You cannot access the website.”);

}

We’re currently verifying whether the client entered the content “letme”. Provided that this is true, the user_name is right; if not, print an alternate message. Attempt it out to see what happens. When you first load the page, before you even click the button, or else you may see the content ” You cannot access the website.” showed over the textbox. That is on account of the fact that we haven’t verified whether the Submit on the structure was clicked.

Working Of Submit Button

In the last section, you perceived how to get content from a textbox when a Submit on a structure was clicked. Then again, when you first load the page, the content still shows. The motivation behind why the content showcases when the page is initially stacked is on account of the script executes whether the button is clicked or not.

This is the issue you confront when a PHP script is in agreement with the HTML, and is constantly submitted to itself in the ACTION property. To get around this, you can do a straightforward check, utilizing an alternate IF Statement. What you do is to check if the Submit was clicked. In the event that it was, go ahead and execute your code. To check if a submit was clicked, utilize the following piece of code:

if(isset($_post[‘submit_1’]) { }

You must think about the if proclamation. Anyhow in the middle of the round sections, we have isset( ). This is an inbuilt capacity that checks if a variable has been set or not. In the middle of the round sections, you write what you need to check using isset( ). For us, this

is $_post[‘submit’]. On the off chance that the client recently revived the page, then no worth will be set for the Submit. In the event that the client did click the Submit button, then PHP will consequently give back a value. Transform your script from the past page to the accompanying and attempt it out:

```
if (isset($_post[‘submit_1’])) {
        $my_user_name = $_post[‘user_name’];
        if ($my_user_name == “letme”) {
                print (“Welcome User!”);
        }
        else {
                print (“You cannot access the website.”);
        }
}
```

ACTION Attribute

You don’t need to submit your structure information to the same PHP page, as we’ve been doing. You can send it to an altogether distinctive PHP page. To perceive how it functions, attempt this:

Make the accompanying page, and call it next_page_2.php. This is your HTML. Notice the ACTION attribute.

```
<html>
<head>
<title>Another HTML Form</title>
```

```
</head>

<body>

<form name =“form_1” Method =“post” Action =“submit_form.php”>

<input TYPE = “text” VALUE =“user_name” Name =“user_name”>

<input TYPE = “Submit” Name = “Submit_1” VALUE = “Login_value”>

</Form>

</body>

</html>
```

Presently make the accompanying page, and call it submit_form.php:

```
<?php

$my_user_name = $_post[‘user_name’];

if ($my_user_name == “letme”) {

        print (“Welcome User!”);

}

else {

        print (“You cannot access this site.”);

}

?>
```

In the PHP script above, notice how there are no HTML labels. Also we’ve forgotten the code that checks if Submit was clicked. That is on account of the fact that there is no PHP left in the first page. The code just gets executed IF the Submit is clicked.

Presenting structure information on an alternate PHP script is an approach to keep the

HTML and PHP separate. However there is an issue with it. The script gets executed on another page. That implies your structure will vanish!

Data Retention

As mentioned previously, at the point when the next_page.php structure is submitted, the points of interest that the client entered get deleted. You're left with the VALUE that was situated in the HTML. For us, user_name continued showing up in the text box when the button was clicked. You can keep the information the client entered effortlessly. Your script ought to now resemble the one in the connection underneath. If not duplicate and glue this script, and test it out on your server.

On the off chance that you take a gander at the VALUE of the text box in the HTML from the above script, you'll see that it's set to "user_name". Since the structure gets presented again on itself, this worth will keep re-showing up in the textbox when the page is submitted. More regrettable, on the off chance that you've left the Value properties discharge, everything the client entered will vanish. This can be extremely irritating, in case you're asking the client to attempt once more.

A better approach is to POST back the values that the client entered. To post the subtle elements once again to the structure, and therefore keep the information the client has officially written out, you can utilize this:

Value="<?php print $user_name ; ?>"

As it were, the VALUE is currently a PHP line of code. The line of code is simply this:

<?php

print $user_name ;

?>

It's a bit hard to peruse, on the grounds that its all on one line.

You additionally need to change your PHP code in the HEAD segment to incorporate an else construct:

```
if (isset($_post[‘submit_1’]))
{
        $user_name = $_post[‘user_name’];
        if($user_name == “letme”) {
        print (“Welcome User!”);
}
else {
    print (“You cannot access this site”);
}
}
else {
    $user_name =””;
}
```

In the else proclamation toward the end, we’re simply setting the value of the variable called $user_name for when the button is NOT clicked, i.e. at the point when the page is revived. Be that as it may, there are some security issues connected with textboxes (and other structure components).

At the same time our new line of HTML for our textbox peruses like this:

```
<input TYPE = “text” Name =‘user_name’ Value=”<?php print $user_name ; ?>”>
```

At the end of the day, we’re presently printing out the VALUE with PHP code.

Radio Buttons

A Radio Button is an approach to limit clients to choose one of the options available. Cases are: Male/Female, Yes/No, or answers to studies and tests. To get the value of a radio button with PHP code, again you get to the NAME trait of the HTML structure components.

In the HTML page, the NAME of the Radio button is the same – “sex”. The main Radio Button has an estimation of “male” and the second Radio Button has an estimation of female. When you’re composing your PHP code, either of these values is returned. Here’s some PHP code to illustrate this concept. You can add this code to the HEAD area of your HTML:

```
<?php

$selected_radio_button = $_post[sex];

print $selected_radio_button;

?>
```

This is pretty much the same code as we utilized for the content box! The main thing that is changed (other than the variable name) is the NAME of the HTML structure component we need to get to – “sex”. The last line simply prints the value to the page. Once more, however, we can add code to discover if the client clicked Submit:

```
if (isset($_post[‘submit_1’])) {

$selected_radio_button = $_post[‘sex’];

print $selected_radio_button;

}
```

Once more, this is the same code you saw before. Simply, get to the structure component

called “Submit_1” and check whether it is situated. The code just executes in the event that it is. Go for the code. Select a radio button and click Submit. The decision you made is printed to the page - either “male” or “female”.

What you will perceive, in any case, when you go for the code is that the dab vanishes from your chosen radio button after Submit is clicked. Once more, PHP is not holding the value you chose. The answer for radio Buttons, however, is somewhat more mind boggling than for simple text boxes

Radio buttons have an alternate characteristic - checked or unchecked. You have to set which button was chosen by the client, so you need to compose PHP code inside the HTML with these qualities - checked or unchecked. Here’s a way you can implement this:

```
<?php

$male_button_status = ‘unchecked’;

$female_button_status = ‘unchecked’;

if (isset($_post[‘submit_1’])) {

$selected_radio_button = $_post[‘sex’];

if ($selected_radio_button = ‘male’) {

$male_button_status = ‘checked’;

}

else if ($selected_radio_button = ‘female’) {

$female_button_status = ‘checked’;

}

}

?>
```

Code for HTML Form

<form name =“form_1” strategy =“post” activity =“radio_button_page.php”>

<input sort = “Radio” Name = “sex” value= “male”

<?php print $male_button_status; ?>

>male

<input sort = “Radio” Name = “sex” value= “female”

<?php print $female_button_status; ?>

>female

<p>

<input sort = “Submit” Name = “Submit_1” VALUE = “Select one of the radio buttons.”>

</Form>

Did we say somewhat more unpredictable? It is substantially more mind boggling than any code you’ve composed till now. Observe the PHP code inside the HTML first:

<?php print $female_button_status; ?>

This is simply a print statement. What is printed out is the value within the variable. What is within the variable will be either the saying “checked” or the statement “unchecked”. Which of these it is relies on upon the rationale from our long PHP at the highest point of the page. How about we separate that.

Initially we have two variables at the highest point of the code:

$male_button_status = ‘unchecked’;

$female_button_status = ‘unchecked’;

These both get set to unchecked. That is just on the off chance that the page is refreshed,

instead of the Submit being clicked.

Next we have our verify whether Submit is clicked:

if (isset($_post[‘submit_1’]){}

This is done in precisely the same manner as done before. As is the following line that puts which radio catch was chosen into the variable:

$selected_radio_button = $_post[‘sex’];

We then need some conditional logic. We have to set a variable to “checked”, so we have an if, else … if development:

if ($selected_radio_button == ‘male’) {

}

else if ($selected_radio_button == ‘female’) {

}

Whatever we’re doing is using what is within the variable called $selected_radio_button. In the event that its “male”, you can do one thing. On the other hand, if its ‘female’, do an alternate thing. The code shown below illustrates how.

if ($selected_radio_button == ‘male’) {

$male_button_status = ‘checked’;

}

else if ($selected_radio_button = ‘female’) {

$female_button_status = ‘checked’;

}

On the off chance that the “male” radio button was clicked, the $male_button_status

variable should be set to ‘checked’. In the event that the “female” alternative was clicked, the $female_button_status variable should be set to ‘checked’.

Working with Checkboxes

Like radio buttons, checkboxes are utilized to give guests options of alternatives to choose from. Though radio buttons confine clients to choose one of the presented options, you can choose more than one choice with checkboxes.

You don’t need the ticks vanishing from the checkboxes, if the client has neglected to enter some different points of interest mistakenly. We saw with Radio Buttons that this can include some dubious coding. The same is valid for checkboxes.

When we coded for the Radio buttons, we gave the buttons the same NAME. That is on account of the fact that only a single alternative can be chosen with radio buttons. Since the client can choose more than one alternative with checkboxes, it bodes well to provide for them distinctive NAME values, and treat them as discrete elements.

In your PHP code, the procedure is to check whether every checkbox component has been checked or not. It’s pretty much the same concerning the radio buttons. In the first place we set up five variable and set every one of them the unchecked, much the same as we did previously for radio buttons:

$check_box_1 = ‘unchecked’;

$check_box_2 = ‘unchecked’;

$check_box_3 = ‘unchecked’;

$check_box_4 = ‘unchecked’;

$check_box_5 = ‘unchecked’;

The following code checks if the submit button was clicked.

```
if (isset($_post[‘submit_1’])) {

}
```

Within this code, on the other hand, we have an alternate isset( ) capacity:

```
if ( isset($_post[‘cb_1’]) ) {

}
```

This time, we’re verifying whether a checkbox was clicked. We have to do this on account of an eccentricity of HTML checkboxes. On the off chance that they are not ticked, they have no value whatsoever, so nothing is returned! On the off chance that you attempt the code without checking if the checkboxes are ticked, then you’ll need to manage a ton of “unclear” blunders.

On the off chance that the checkbox is ticked, however, it will give back a value. Thus, the isset( ) capacity will be true and the if code will be executed.

```
if($cb_1 == ‘val_1’) {

$cb_1 = ‘checked’;

}
```

This is yet an alternate If statement! Anyhow, we’re simply checking the estimation of a variable. We have to comprehend what is within it. This one says, “If the quality within the variable called $cb_1 is “value_1” then execute some code. The code we have to execute is to put the content “checked” within the variable called $cb_1. Whatever remains of the if articulations are the same – one for every checkbox on the structure. The exact opposite thing we have to do is to print the value of the variable to the HTML structure:

```
<input sort = “checkbox” Name =‘cb_1’ value =“value_1”
```

<?php print $cb_1; ?>

>visual Basic .NET

Once more, this is the same code you saw with the radio buttons. The PHP part is:

<?php print $cb_1; ?>

So we’re simply printing what is within the variable called $cb_1. This will either be “checked” or “unchecked.” There are other answer for checkboxes, yet none appear basic! The point here, however, is that to take care of the concepts you learnt in the chapter on conditional logic.

Loops in PHP

So what's a loop then? A loop is something that goes all around. In computer programs, it's precisely the same. But a programming loop will go all around until you let it know where to stop. You additionally need to tell the system two different things - where to begin your circle, and what to do after it's done one lap, which is also known as the upgrade declaration. You can program without utilizing loops. Anyway, using them makes your code compact and much simpler to comprehend.

In order to understand the concept of loops, consider the following example:

Consider a code where you need all numbers from 1 to 5. The answer to this coding problem is simply this:

```
$answer_val = 1 + 2 + 3 + 4 + 5;

print $answer_val;
```

Genuinely basic, you think. Also very little code, either. However consider the possibility that you needed to include a thousand numbers. Is it accurate to say that you are truly going to sort every one of them out like that? It's a horrendous parcel of writing. A loop would make life a ton less complex. You utilize them when you need to execute the same code again and again.

We'll examine a couple of kinds of programming loop in this chapter. However, the For Loop is the most utilized kind of loop, and this is why we will discuss the for loop first.

For Loops

For loop is the most commonly used looping construct in PHP. It is usually preferred over its counterparts because of its easy implementation and hassle-free comprehension. Here's a PHP For Loop in a little script. Add it into a new PHP script and save your work. Then

run your code and test it out.

```
<?php

$start_val = 1;

$count_val = 1;

for($start_val; $start_val <= 10; $start_val++) {

print $count_val . “<br>”;

$count_val = $count_val + 1;

}

?>
```

How could you have been able to you get on? You ought to have seen the numbers 1 to 10 printed on your program page.

The general syntax for a for loop is:

```
for (starting value; ending value; update value expression) {

}
```

The principal thing you have to do is sort the name of the loop you’re utilizing, which for this situation is for loop. In the middle of round brackets, you then place your three conditions:

Begin Value

The principal condition is the place you tell PHP the introductory value of your loop variable. In order, what should PHP start the loop with? We utilized this:

```
$start_val = 1;
```

We’re allocating an estimation of 1 to a variable called $start_val. Like all variables, you

can make up your name of the variable. A prominent name for the introductory variable is the letter i or j. You can set the starting condition before the loop starts, as we did:

$start_val = 1;

for($start_val; $start_val <= 10; $start_val++) {

On the other hand you can relegate your loop variable value right in the ‘For Loop’ code, as shown below:

for($start_val = 1; $start_val <= 10; $start_val++) {

The result is the same as the beginning number for this loop is 1.

End Value

Next, you need to advise PHP when to end your loop. This can be a number, a string, a Boolean value and any other expression or condition. Here, we’re advising PHP to continue going round the loop while the estimation of the variable $start_val is lesser than or equal to 10.

for($start_val = 1; $start_val <= 10; $start_val++) {

At the point when the estimation of $start is 11, PHP will end the loop and come out of the scoping brace.

Upgrade Expression

Loops require a method for getting the next number for looping variable, which is given in the form of an update expression in the for structure. On the off chance that the loop couldn’t upgrade the beginning value, it would be stuck on the beginning value. As it were, you have to advise the circle how it is to go all around. We utilized this:

$start_val++

In a ton of programming languages (and PHP), the twofold or more operators (++) implies

addition (expand the quality by one). It’s simply a short method for saying this:

$start_val = $start_val + 1

You can decrement the value in the same manner by utilizing the twofold less operator (- -), yet we won’t go into that.

So our entire loop peruses “Beginning at an estimation of 1, continue going all around while the value of the looping variable is short of 11 and update the beginning value by one each time the loop is executed.”

Each time the loop goes round, the code between our two wavy sections { } gets executed:

print $count_val . “
”;

$counter = $count_val + 1;

Perceive that we’re simply augmenting the counter variable by 1 each one time round the circle, precisely in the same way as we are doing with the start_val variable. So we could have put this:

$count_val ++

The impact would be the same. As a trial, have a go at setting the estimation of $count_val to 11 outside the circle, which is at present $count_val = 0. At that point inside the circle, use $counter- - (the twofold less sign). Could you think about what will happen? Will it crash, or not? On the other hand will it print something out? Better save your work, in the event that something goes wrong!

While Loop

As opposed to utilizing a for circle, you have the choice to utilize a while loop. The structure of a while loop is more basic than a for loop, on the grounds that you’re just assessing one condition. The loop goes all around while the condition is true. At the point when the condition is false, it breaks out of the whole circle. Here’s the syntax:

```
while (testing_condition) {

//Code

}
```

The following code illustrates the implementation of a while loop.

```
$count_val = 1;

while ($count_val < 11) {

        print (" count_val = " . $count_val . "<br>");

        $count_val++;

}
```

The condition to test for is $count_val < 11. Each one time round the whole circle, that condition is checked. In the event that count_val is short of eleven, then the condition is true. At the point when $counter is more noteworthy than eleven then the condition is false. A while circle will quit going all around when a condition is false.

In the event that you utilize a while circle, be cautious that you don't make an infinite loop. You'd make one of these on the off chance that condition yields a true in all cases.

Do-While Loop

This sort is loop is very nearly indistinguishable to the whole circle, aside from the fact that the condition has a go at toward the end:

```
do

{

//Code

}while (testing_condition)
```

The distinction is that your code gets executed in any event at least once. In an ordinary while circle, the condition could be met before your code gets executed and the loop will be exited right away. However, here the condition is checked after the loop code has executed once.

Don’t stress excessively over do … while loops. Focus on for loops for practical programming purposes. At the same time, there is an alternate kind of loop that proves to be useful, which is also called the For Each loop. However, it is rarely used and thus we have chosen to skip its description altogether.

Break Keyword

There are times when you have to break out of a loop before the entire thing gets executed. Alternately, you need to break unaware of what’s going on in light of a lapse that your code or client may have made. In such a case, you can utilize the break keyword. Luckily, this includes just writing break followed by a semicolon.

Arrays

By now, it is imperative that you comprehend what a variable is – simply a stockpiling territory where you hold numbers and text. The issue is, a variable will hold one and only one value at a time. You can store a solitary number in a variable, or a solitary string. An array is similar to any unique variable, which can hold more than one number, or more than one string, at once. On the off chance that you have a list of things (like client requests, for instance), and you have to do something with them, then it would be very awkward to do this:

$order_1 = “Shoes Variety 1”;

$order_2 = “Shoes Variety 2”;

$order_3 = “Shoes Variety 3”;

$order_4 = “Shoes Variety 4”;

You may use a loop through your requests and discover a particular one or access records and make changes to them one by one. What’s more, imagine a scenario where you had not four requests, but you have more than four hundred of these. A solitary variable is plainly not the best programming apparatus to use here. In this case, an array is one of the more practically feasible storage options for you. An array can hold all your requests under a solitary name. Furthermore you can get to the requests by simply alluding to the array name followed by the index of the concerned request.

In order to set up an array, you can use the following syntax:

$order_array = array( );

To start with you write out what you need your cluster to be called ($order_array, in the example shown above) and, after an equivalents sign, you write this:

array( );

So, setting up an array simply includes writing the expression array, followed by a couple of round brackets. This is sufficient to advise PHP that you need to set up the array. Be that as it may, there's nothing in the cluster yet. Whatever we're doing with our line of code is advising PHP to set up an array, and provide for it the name $order_array.

Adding Elements To The Array

You can utilize two essential strategies to place something into an array.

Strategy One – Type between the round sections

The primary strategy includes writing your values between the round sections of array(). In the code beneath, we're setting up a cluster to hold the types of coats in your wardrobe:

$coat_types = array("Winter Coats", "Summer Coats", "Short Coats", "Over Coats");

So the name of the array is $coat_types. Between the round sections of array(), we have written a few values. Each one value is differentiated by a comma.

Arrays work by having a position, and some information for that position. In the above array, "Winter Coast" is in position zero, "Summer Coats" is in position 1, "Short Coats" is in position 2, and "Over Coats" is in position 3.

The main position is constantly zero, unless you tell PHP otherwise. Anyhow the position is know as a key. The key then has a worth connected to it. You can detail your own particular numbers for the keys in the following manner:

$coat_types = array(1 => "Winter Coats", 2 => "Summer Coats", 3 => "Short Coats", 4 => "Over Coats");

So, you write a number for your key, emulated by the equivalents sign and a right plot section (=>). In the exhibit over, the first Key is currently 1 and not 0. The thing put away under key 1 is "Winter Coats". The last key is 4, and the thing put away under key 4

is “Over Coats”. Watchful of every one of the commas, when you set up an array like this.

Technique two – Assign values to a cluster

An alternate approach to place values into an array in a manner similar to this:

$coat_types = show();

$coat_types[ ]=“Winter Coats”;

$coat_types[ ]=“Summer Coats”;

$coat_types[ ]=“Short Coats”;

$coat_types[ ]=“Over Coats”;

Here, the array is initially created with $coat_types = array();. This advises PHP that you need to make an array with the name of $coat_types. To store values in the exhibit you first sort the name of the cluster, emulated by a couple of square sections:

$coat_types[ ]

After the equivalents sign, you write out what you need to store in this position. Since no numbers were written in the middle of the square sections, PHP will allot the number 0 as the first key:

0=> “Winter Coats”,

1=> “Summer Coats”,

2=> “Short Coats”,

3=> “Over Coats”

Accessing Array Elements

Actually, there are few ways you can do it. Yet the “Key” is the key. Here’s an illustration for you to attempt:

```
<?php
$coat_types = array(“Winter Coats”, “Summer Coats”, “Short Coats”, “Over Coats”);
print $coat_types[0];
?>
```

The array is the same one we set up some time recently. To get at what is within a cluster, simply sort the key number you need to get to. In the above code, we’re printing out what is held in the 0 position (Key) in the array. You must simply place the key number between the square sections of your show name in the following manner:

```
print $array_name[0];
```

Associative Arrays

Your clusters keys don’t need to be numbers, as in the past segment. They can be strings as well. This can help you recollect what’s in a key, or what it should do. When you utilize content for the keys, you’re utilizing an associative cluster; when you utilize numbers for the keys, you’re utilizing a scalar array. Here’s a cluster that sets up first name and surname consolidations:

```
$name = array( );
$name[“nick”] = “Carter”;
$name[“richard”] = “Waters”;
```

To get to the values in an associative array, simply allude to the Key name:

```
print $name[“nick”];
```

Notwithstanding, on the grounds that associative arrays don’t have numbers for the keys, an alternate system is utilized to loop around them – the For Each loop.

Sorting Arrays

There may be times when you need to sort the values within an array. Case in point, assume your array elements are not in order. Like this one:

$name = array( );

$name[“nick”] = “Carter”;

$name[“richard”] = “Waters”;

To sort this array, you simply utilize the asort( ) function. This includes nothing more intricate than writing the statement asort, emulated by round sections. In the middle of the round sections, sort for the sake of your associative:

asort($name);

Different functions that can be utilized to sort values in shows include:

rsort( ) – Sorts a scalar array in converse request

arsort( ) - Sorts the values of an associative cluster in opposite request

krsort( ) - Sorts the keys of an associative cluster in opposite request

Getting Random Keys

You can get a random key from a cluster. This could be valuable in some coding scenarios.

<?php

$number_array = array(1 => 1, 2 => 2, 3 => 3, 4 => 4, 5 => 5, 6 => 6);

$random_key_element = array_rand($number_array, 1);

print $random_key_element;

?>

The function that furnishes a proportional key is this:

array_rand($number_array, 1);

You begin off with the function array_rand( ). In the middle of the round sections, you require two things: the name of your array, and what number of keys you need.

The Count Function in PHP

The count( ) function is valuable when you need to return what number of components are there in your array.

To get what number of components are in the cluster, we utilized this:

$array_count = count($coat_types);

The function take strings of content and control them is one of the key capacities you require as a developer. In the event that a client enters subtle elements on your structures, then you have to check and accept this information. Generally, this will include doing things to this text.

Some examples of these manipulations include changing over letters from lowercase to uppercase or vis-a-versa, checking which program the client has or cutting down on white space from content entered in a content box. These go under the heading of string control. To make a beginning, we'll take a gander at changing the case of character.

Changing the Case of a Character

Assume a you have a textbox on a structure that asks clients to enter a first name and surname. The chances are high that somebody will enter this:

jack waters

Rather than this:

Jack Waters

So your work as a developer is to change over the first letter of each one name to uppercase from lower. This is simple, with PHP. It's simply a textbox and a button. The textbox will have "jack waters" entered, when you stack it up. What we need to do is to transform it to "Jack Waters" when the button is clicked. Here's the script that does that.

```
<?php

$name = 'jack waters';

if(isset($_post['submit_1'])) {
```

```
        $name = $_post[‘user_name’];

        $name = ucwords( $name );

}

?>
```

The primary line simply verifies that the lowercase adaptation is put into the textbox when the page loads:

$name = ‘jack waters’;

This is the line that we need to change over and turn into” “Jack Waters”. The main line in the code that you haven’t yet met is this one:

$name = ucwords($name);

What’s more, that is everything you need to change over the first letter of each saying to uppercase. The inbuilt function that is used for this purpose is:

ucwords( )

In the middle of the round sections, you need to write the variable or content you need to change over. PHP will deal with the rest. At the point when the change is finished, we’re putting it away into the variable called $name.

In the event that you simply need to change over the first letter of a string (for a sentence, for instance), then you can use ucfirst( ) .

Trimming White Space

Something else you’ll need to do is to trim the white (clear) space from content entered into textboxes. This is simple, as there’s some helpful PHP inbuilt-functions to help you do this. Assume your client has entered this in the textbox:

” user_name ”

From the quotes, we can see that there is additional space in the text, before and after the main content. We can count what number of characters this string has with an alternate valuable capacity: strlen( ). As its name proposes, this gives back where its due of a string. By length, we mean what number of characters a string has. Attempt this script:

```
<?php

$w_space = ” user_name “;

$count_val = strlen($w_space);

print $count_val;

?>
```

When you run the script, you’ll find that the variable contains 15 characters. Notwithstanding, user_name has just 9 characters. In case you’re checking for a precise match, this matters!

To uproot the white space, you can utilize the trim( ) capacity. Change your script to this:

```
<?php

$w_space = trim(” user_name “);

$count_val = strlen($w_space);

print $count_val;

?>
```

When you run the script now, you ought to find that the variable has the right number of characters - 9. That is on account of the trim( ) capacity expels any clear spaces from the left and right of a string. Two related functions are ltrim( ) and rtrim( ). The first, ltrim( ), expels space from the earliest starting point of a string; the second one, rtrim( ), expels space from the end of a string.

The strpos Function

A more helpful thing you'll need to do is to check whether one string is within an alternate. Case in point, you can get which program the client has with this:

$client = $_server[“http_user_client”];

print $client;

Attempt it out and see what gets printed out. You ought to find that this string gets printed. In case you're trying which program the client has, you can utilize a string function to look for a short string within this long one. A PHP string function you can utilize is strpos( ). The sentence structure for the strpos function is:

strpos(string_to_consider, string_to_find, begin)

Part Lines

To part lines of content, the explode ( ) function can be utilized. You recently gave it the content you need to part, and the character that is utilized to discrete each one piece. Here's the sentence structure:

explode(separator, string_to_split)

In the middle of the round sections of explode( ), the separator you need to utilize goes initially, emulated by a comma, then the string you need to part. For our line of code above, you'd do this:

$line_text = “Numbers 1, 450, 3500, 264, 500, 7869”;

$line_text = explode(“,” , $line_text);

So basically we're stating, “search for a comma in the content, and part the line of content into particular pieces.” Once PHP does its employment, it puts all the parts into the variable on the left hand side of the equivalents sign (=), which was $line_text for us.

This variable will then be an array!

Joining Strings

In the event that you have a line of content in a array, you can go along with everything together to structure a solitary line of content. This is the polar opposite of explode. This time, make use of implode( ):

$coat_type = array(“Winter Coats”, “Summer Coats”, “Short Coats”, “Over Coats”);

$line_new = implode(“,”, $coat_type);

Here, we have an array called $coat_type. The content in the array needs to be joined before keeping in touch with it again to a content record. The implode( ) function does the joining. The structure for the implode( ) function is similar to that of explode( ).

implode(separator, texts_to_join)

So, implode( ) will join all the content together and separate each one section with a comma, in the code above. Obviously, you don’t need to utilize a comma as a rule. You could utilize whatever other character.

Escape Characters in PHP

Escape sequence in PHP doesn’t mean breaking free and “doing a runner”. It is a method to keep PHP from terminating your strings too soon, or for verifying you have the right string data returned. Here’s an illustration. Attempt this script:

<?php

$my_string = ‘This is Sarah’s car’;

print $my_string;

?>

Verify you write the script precisely as it is, with all the single quote imprints. Next, run

the script. What you ought to discover is that PHP provides for you a message of error. The reason is that you have three single quote imprints. PHP gets confounded, in light of the fact that it doesn’t realize what your string is. To tackle the issue, you could utilize twofold quotes on the outside. Like this:

$my_string = “This is Sarah’s car”;

Then again you could get away from the punctuation. You get away from a character by writing a “\” before it. Like this:

$my_string = ‘This is Sarah\’s car’;

On the off chance that you attempt that out, you ought to find that the string prints perfectly.

Now attempt this script:

<?php

$new_string = ‘mypath';

print $new_string;

?>

Once more, you’ll get a message from PHP citing an error in your code. Encompass it with twofold quotes rather than single quotes and run the script once more. Does the string print? The reason it doesn’t is on account of you haven’t gotten away from the escape character. PHP sees it as an uncommon character, and is expecting more subtle elements after the back-slash. In order to avoid such a lapse, you can use the following code:

$new_string = ‘mypath\’;

So now we have two cuts on the end of the string. When you run the script, you ought to observe that it prints out this:

mypath\

In the event that your PHP script is not giving back it’s due that it is ought to do, then you may need to utilize the slice to escape them. You additionally need to escape certain characters when working with databases, else, you’re opening yourself up to a long list of error messages and warnings.

Some Important String Functions

As opposed to enumerating all the conceivable string functions you can utilize, we’ll simply provide for you a concise list. There’s a case of how to utilize each one string function, in the event that you click on the connections beneath. Simply make use of them as and when required.

ord( ): determines a character’s ASCII value

chr( ): performs conversion of ASCII value to character

similar_text( ): determines the similarity between two text strings

echo( ): used as an alternative to print

substr( ): returns a substring from the string

str_replace( ): performs replacement of one string from another

str_repeat( ): performs repetition of characters a specified number of times

strlen( ): returns the string length

str_word_count( ): returns the number of words in a string

Functions in PHP

A function is simply a section of code. However, it is different from your main code. You separate it on the grounds that its decent and helpful, and you need to utilize it not once but again and again. It's a piece of code that you think is valuable, and need to utilize once more. Functions spare you from composing the code again and again.

Assume you have to check content from a textbox. You need to trim any clear spaces from the left and right of the content that the client entered. So on the off chance that they entered this:

" Jack Waters "

You need to transform it into this:

"Jack Waters"

In any case you additionally need to check if the client entered any content whatsoever. You don't need the textbox to be totally clear. You can utilize the PHP inbuilt capacity called trim( ) for solving this purpose.

$entered_text = trim($_post['text_1']);

That will dispose off the white space in the text box. Anyhow it won't check if the content box is clear. You can include an if articulation for that:

if ($entered_text == "") {

error_message = "Nothing written in the text box";

}

Yet suppose it is possible that you have loads of textboxes on your structure. You'd need to have heaps of if proclamations, and check each one single variable for a clear string.

That is a ton of code to compose! Instead of doing that, you can make a solitary function, with one if articulation that can be utilized for each one clear string you have to check.

Utilizing a function means there’s less code for you to compose. What’s more, its more effective. We’ll perceive how to compose a function for the above situation in a minute. Anyhow first and foremost, here’s the syntax structure for a function.

```
function function_name() {

}
```

So you begin by writing the statement function. You then need to think of a name for your function. You can call it very nearly anything you like. It’s much the same as a variable_name. Next, you write two round sections ( ). At long last, you require the two wavy sections also { }. Whatever you do goes between the wavy sections. Here’s a straightforward sample that simply print something out:

```
function print_message() {
        print “Error!”;
}
```

In the case above, we’ve begun with the word function. We’ve then called this specific function print_message().In the middle of the wavy sections, there a print articulation. Attempt it out with this script:

```
<?php
function print_message() {
        print “Error!”;
}
?>
```

Run your script and see what happens. You ought to find that nothing happens! The reason that nothing happened is on account of a capacity is a different bit of code. It doesn't run until you let it know to. Simply stacking the script won't work. It's similar to those inbuilt capacities you utilized, for example, trim.

You can't utilize trim( ) unless you write out the name, and what you need PHP to trim. The same applies to your own particular functions – you need to "tell" PHP that you need to utilize a function that you composed. You do this by just writing out the name of your function. This is known as "calling" a function. Attempt this new form of the script.

```
<?php
function print_message() {
        print "Error!";
}
print_message();
?>
```

After the capacity, we've written out the name once more. This is sufficient to advise PHP to run our code portion. Presently, change your code to this, and see what happens:

```
<?php
print_message();
function print_message() {
        print "Error!";
}
?>
```

On the off chance that you have PHP 4 or above, you ought to see no distinction – the

function will in any case get executed with the name above or underneath the function. Yet for tidiness and coherence's purpose, it's better to put the greater part of your function either at the top or base of your scripts. Then again exceeding all expectations even further, in a different PHP document. You can then utilize an alternate inbuilt function called "include."

Function Scope

There's a thing called scope in programming. This alludes to where in your scripts a variable can be seen. In the event that a variable can be seen from anyplace, it's said to have global scope of functioning. In PHP, variables within capacities can't be seen from outside of the capacity. What's more, functions can't see variables in the event that they are not piece of the capacity itself. Attempt this variety of our script as a case:

```
<?php
$err_message = "Error Found!";
print_message();
function print_message( ) {
        print $err_message;
}
?>
```

This time, we have set up a variable called $err_message to hold the content of our message. This is situated up outside of the function. Run the script, and you'll get a PHP error message about " Undefined variable".

In like manner, attempt this script:

```
<?php
```

```
print_message();

print $err_text;

function print_message() {

        $err_text = “Error!”;

}

?>
```

This time, the variable is inside the function. However, we’re attempting to print it from outside the function. Regardless, you get a blunder message. Here’s a right form:

```
<?php

print_message();

function print_message() {

$err_text = “Error!”;

print $err_text;

}

?>
```

Here, we have both the variable and the print explanation set up within the capacity. The mistake message now prints.

So on the off chance that you have to inspect what is within a variable, you require an approach to get the variable to the capacity. That is the place contentions come in.

Arguments

Functions can be given variables, with the goal that you can do something with what’s within them. You ignore the variable to your function by writing them within the round

sections of the function name. Here’s a script like the one you saw prior:

```
<?php

$err_text = “Error!”;

print _message($err_text);

function print_message($err_text) {

print $err_text;

}

?>
```

The main contrast is the that we now have something between the round sections of our function:

```
function print_message($err_text) {

}
```

The name is the same, yet we’ve put a variable in the middle of the round sections. This is the variable that we need to do something with. The one called $err_text. By writing a variable within the round sections, you are setting up something many refer to as an argument. An argument is a variable or value that you need your capacity to manage.

Calling Functions

```
$err_text = “Error!”;

print_message($err_text);
```

The first line places something into the variable. Anyhow, when you need to hand something to a function that has an argument, you have to write it in the function call. In our script, we’re writing the name of the variable.

However, if you attempt to call this function without giving any argument, like shown below, you can expect to get an error message from PHP.

print_message( );

That is letting you know that your function has been set up to take an argument, yet that you've left the round sections void when you attempted to call the function. Your functions can have more than 1 arguments. Simply separate every argument with a comma. A sample implementation of this sort is given below:

function check_for_errors($err_text, err_flag) {

}

Getting Values From Functions

When you're making your function, you may recognize that they can be softened down up to two classes: functions that you can leave, and simply let them do their tasks; and functions where you have to recover an answer. As a sample, here's the two separate classes in activity:

print ("Message!");

$str_len = strlen($str_len);

The print function is a sample of a function that you can leave, and simply let it do its tasks. You simply let it know what to print and it gets on with it for you. On the other hand, a function like strlen( ) is definitely not. You need something back from it – the length of the string.

Assume you had a function that worked out a 20 percent markdown. Anyhow, you just need to apply the markdown if the client used in excess of 100 pounds. You could make a function that is given the sum used. At that point verify whether its over a 100 pounds. In the event that it is, the function ascertains the rebate; if not, don't worry about the

markdown. In both cases, you need the function to give back where its due to your inquiry – What do I charge this client? Here’s the script:

```
<?php

$t_expenditure = 200;

$total_order = compute_order_total($t_expenditure);

print $total_order;

function compute_order_total($t_expenditure) {

$discount = 0.2;

if ($t_expenditure > 100) {

$t_discount = $t_expenditure - ($t_expenditure * $discount);

$t_charge = $t_discount;

}

else {

$t_charge = $t_expenditure;

}

return $t_charge;

}

?>
```

The lines to focus on are the ones for the $t_expenditure variable. The code first sets up an aggregate sum used, which in practice may originate from a structure on a content box, or a shrouded field:

```
$t_expenditure = 200;
```

The following line is our function call:

$total_order = compute_order_total($t_expenditure);

The function call is currently on the right of the equivalents sign (=). To the left of the equivalents sign is simply an ordinary variable - $total_order. In case you’re setting up your function like this, then you are asking PHP to give back a value from your functions, and put the answer into a variable on the left of the equivalents sign.

PHP will go off and figure your function. When it discovers an answer, it will attempt to give back a value. The answer will be put away for the sake of your function, compute_order_total( ) for us. Anyhow take a gander at the function itself, and the line toward the end:

function compute_order_total($t_expenditure) {

$discount = 0.2;

if($t_expenditure > 100) {

$t_discount = $t_expenditure - ($t_expenditure * $discount);

$t_charge = $t_discount;

}

else {

$t_charge = $t_expenditure;

}

return $t_charge;

}

The last line is:

return $t_charge;

The return word advises PHP to give back a value. The value it returns is whatever you have put away in the variable that comes after the statement return. Here, were advising PHP to situated the response to the function called compute_order_total( ) to whatever is put away in the variable we've called $t_charge. It's this that will get put away in our variable called $total_order.

In case you are discovering this a bit precariously, recall what a function is: a different bit of code that does some work for you. It can either give back a value, or not give back a value. It depends completely on your needs.

Calling a Function by Reference or Value

Functions can be difficult to use, in the event that you've never utilized them previously. An alternate troublesome part to comprehend is the manner by which values can change, or not change, contingent upon extension. Scope, on the off chance that you review, alludes to where in your code a variable can be seen. On the off chance that you simply do this, for instance:

```
$var_val = 50;

case( );

function case() {

print $var_val;

}
```

In place for the function to have the capacity to see what's within the variable called $var_val, you can set up the function to acknowledge an argument. You'd then sort the variable name between the round sections, when you call it. This has been illustrated in the following code:

```
<?php
$var_val = 25;
ex_func($var_val);
function ex_func($var_val) {
print $var_val;
}
?>
```

In the event that you run the code above, it now prints out the number twenty five. At the same time, it is critical to understand that you are simply giving the function a duplicate of the variable. Any changes made to the code shall not affect the variable in the code of the calling function. As a case, change your code to this:

```
<?php
$var_val = 25;
print “Before the function call = ” . $var_val . “<br>”;
ex_func($var_val);
print “After the function call = ” . $var_val;
function ex_func($var_val) {
$var_val = $var_val + 25;
print “Within the function = ” . $var_val . “<br>”;
}
?>
```

Here, we have three print structures: one preceding the call to the function, one within the

function, and one after the function call. Yet we're printing out the estimation of the variable called $var_val each one time. Within the function, we're adding 25 to the estimation of the variable. When you run the code, it will print out this:

Prior to the function call = 25

Within the function = 50

After the function call = 25

The critical one is after the function call. Despite the fact that we changed the estimation of $var_val within the function, regardless it print 25 after the function call! That is on account of the capacity was given a duplicate, and NOT the first.

When you hand a function a duplicate of a variable, it is called passing the variable by value (simply a duplicate). The option is to NOT pass a duplicate, however to allude once more to the first. Roll out one little improvement to your script. This is the sample code for this scenario.

function ex_func(&$var_val) {

The main expansion is a & character before the variable between round sections. This advises PHP that you need to roll out improvements to the first, and don't simply need a duplicate. When you run the script, it now print out the accompanying:

Prior to the capacity call = 10

Within the capacity = 20

After the capacity call = 20

After the capacity call, we now have an estimation of 20! So a change to the estimation of the variable outside the capacity has been made. When you rolls out improvements to the first like this current, its called passing the variable by reference

Server Variables in PHP

PHP stores data about the server. This will entail things like, the program the guest is utilizing, the IP location, and which site page the guest originated from. Here's a script to attempt with those three server variables:

$browsing_program = $_server[‘http_user_agent’];

$referring_agent = $_server[‘http_referring_agent’];

$ip_address = $_server[‘remote_address’];

print “IP Adress = ” . $ipaddress;

print “Program = ” . $browser . “
”;

print “Referrer = ” . $referrer . “
”;

These are valuable in the event that you need to log your details, or to boycott a specific IP address. On the off chance that you run the script on a neighbourhood machine, you may get an error message for the referrer.

So to get at the qualities in server variables, the sentence structure is this:

$_server[‘server_variable’]

You begin with a dollar sign, then an underscore character ($_). At that point you include the expression SERVER. In the middle of square sections, you write the name of the server variable you need to get to. Encompass this with either single or twofold quotes.

Since you are giving back a value, you have to put all that on the right hand side of an equivalents sign. On the left of the equivalents sign (=), you require a variable to hold the string that is returned. The server variables are held in an array, so you can utilize a foreach circle to get a list of all accessible ones.

Header Function

When you ask for a site page to be brought once more to your program, you're not simply bringing back the page. You're likewise bringing back something many refer to as a HTTP HEADER. This is some additional data, for example, kind of system making the appeal, date asked for, if it be shown as a HTML report, to what extent the archive is, and much more.

One of the things HTTP HEADER likewise does is to give status data. This could be whether the page was discovered (404 errors), and the area of the archive. In the event that you need to redirect your clients to an alternate page, here's a sample:

<?php

header("location: http://www.homeandlearn.co.uk/");

?>

<html>

<body>

</body>

</html>

Note how the header code goes before any HTML. In the event that you put header code after the HTML, you'll get an error message along the lines of "header data cannot be changed."

Include Function

Having the capacity to incorporate different documents into your HTML code, or for your PHP scripts, is a valuable thing. The include( ) function permits you to do this. Assume you have a document that you need to incorporate in an online page. You could duplicate and glue the content from the record straight into you HTML. On the other hand, you can also utilize the include( ) function for the same purpose.

The capacity to open up records, for example, plain text or CSV documents is an extraordinary resource for you as a developer. Not every task obliges a database with various tables, and put away straightforward information in a record can be a decent option. This is true particularly if your web host doesn't permit you to have a database.

Opening A File

To open up a record, there are a couple of techniques you can utilize. The one we'll begin with is readfile( ). As its name propose, it peruses the content of a file or record for you. Attempt this basic script for an introduction to file handling in PHP.

```
<?php

$doc_value = readfile( "my_dictionary.txt" );

print $doc_value;

?>
```

The readfile( ) function is valuable if all you need to do is open up a record and read its content.

An alternate function that simply peruses the substance of a record is file_get_contents( ). It is accessible in PHP rendition 4.3 or more. Here's a case:

```
<?php

$file_for_reading = "my_dictionary.txt";

print file_get_contents( $file_for_reading );

?>
```

This is utilized as a part of pretty much the same path as the readfile function. The

distinction for us is the change of name to file_get_contents( ). A finer strategy to open records is with fopen( ). This function provides for you more alternatives that, for example, setting whether the record is for perusing, for keeping in touch with too, and a couple of more choices. Here's a case:

```
<?php

$doc_value = fopen( "my_dictionary.txt", "r" );

print $doc_value;

fclose($doc_value);

?>
```

Run this script and see what happens. You ought to see something like the accompanying printed out:

Resource ID #2

Not exactly what you were anticipating! The reason is that fopen( ) doesn't really perused the content of a record. Everything it does is to situate a pointer to the record you need to open. It then returns what's called a document handle. Whatever you're doing is advising PHP to recollect the area of the record.

The "r" on the end signifies "open this petition for perusing only". We'll see different alternatives in a minute. Yet, now that you've advised PHP to recollect the area of the record you need to open, how would you read the content of the document?

One path is to utilize fgets( ). This will read an indicated number of character on a solitary line of content. It's regularly used for looping and reads each one line of content. When you're utilizing fgets( ), you likewise need to check when the end of the document has been arrived at. This is finished with the inbuilt capacity feof - record, end of document.

```
$file_handler = fopen("my_dictionary.txt", "r");
```

In addition, there are different alternatives. The value ‘r’ can be used for read-only, ‘w’ for write-only, ‘r+’ and ‘w+’ for read and write, ‘a’ for append and ‘a+’ for append and read.

Checking If The Record Exists

It’s a decent thought to check if the record exists, before attempting to do something with it. The file_exists( ) function can be utilized for this:

```
if( file_exists( “my_dictionary_2.txt” ) {

print “This file exists!”;

}

else {

print “This file does not exist.”;

}
```

To begin with, we ask PHP to open the record and make a document handle:

```
$file_handler = fopen(“test_file.txt”, “w”);
```

So we’re asking PHP to make a document handle that indicates a content record called “test_file.txt”. In the event that a document of this name can’t be discovered, then one will be made with this name. After a comma, we’ve written “w”. This tells PHP that the document will be opened for writing only.

The third line is the place we manipulate the record:

```
fwrite( $file_handler, $doc_value);
```

In the middle of the round sections of fwrite( ), we’ve set two things: the document we need to keep in touch with, and the content of the record. Also, with the exception of shutting the document, that is everything you need!

file_put_contents( )

On the off chance that you have PHP 5, you can utilize the new function called file_put_contents( ) rather than fwrite( ).

It is used in the same manner as file_put_contents() with the only difference of a third parameter.

file_put_contents($file_handler, $doc_value, setting);

The setting choice can be File_append and File_use_include_path.

So to attach to the record, simply do this:

file_put_contents($file_handler, $doc_value, File_append);

Perusing a Text File And Saving It In An Array

There is an alternate choice you can use to place lines of content into a cluster. In the strategy beneath, we're utilizing the explode( ) string to make an array from each one line of content. Here's the code:

```
<?php

$file_handler = fopen("my_dictionary.txt", "rb");

while (!feof($file_handler) ) {

$text_lines = fgets($file_handler);

$parts_of_line = explode('=', $text_lines);

print $parts_of_line[0] . $parts_of_line[1]. "<br>";

}

fclose($file_handler);

?>
```

Document Locations

There are a couple of inbuilt PHP functions you can use to discover document ways. This is helpful for discovering the careful area (relative or supreme) of your scripts or pages. Before you attempt these out, make another PHP page and save it as file_dir.php.

The following code can be used for determining the path of a file.

```
<?php

$absolutepath = realpath(“file_dir.php”);

print “Path: ” . $absolutepath;

?>
```

To get the accurate way of document, then, you can utilize real_path(). In the middle of the round sections of the function, sort the name of the record. The following code is used to get the directory, yet not the document name.

```
<?php

$dir_name = dirname(“folder/myphp/file_dir.php”);

print “Index is: ” . $dir . “<br>”;

?>
```

In order to get the names of the directory, you can utilize the function dirname( ). This will strip off the name of the record and furnish a proportional payback of the content between the round sections of the capacity.

However, if you wish to get the name of the file, you can use te following code:

```
<?php

$base_name = basename(“folder/myphp/file_dir.php”);

print “Name of the document is: ” . $base_name . “<br>”;
```

?>

If you have to get the name of the document, then utilize the function basename( ). When you write a more drawn out record way in the middle of the round sections of the capacity, it will strip off the rest and leave the name of the document.

Time and Date Functions In PHP

Knowing how to handle time and date values in PHP will be a valuable expansion to your programming abilities. In this and the accompanying segments, we'll investigate how to process this sort of information.

The date( ) function

The inbuilt PHP function date( ) is the most generally utilized strategy for returning date values. Sadly, there is a long rundown of things you can put between the round sections of the capacity! Attempt this script, to get a thought of how it functions:

```
<?php

$day_today = date('d-m-y');

print $day_today;

?>
```

It ought to print the day of the week first (d), then the month (m), then the year (y). Be that as it may, this will be the numerical arrangement. So it will print something like:

21-10-2014

An alternate helpful date/time function is getdate( ). This will give back an associative array with all the time and date values. You can utilize it for things like contrasting one date with an alternate. For instance, looking at how long have passed since a given date. Here's the syntax for using this function:

getdate(time_stamp); The time stamp is non-compulsory. In the event that you forget it, it gets the values for the current neighbourhood time and date.

Since getdate gives back an associative array, you can simply do this kind of thing:

```
$day_today = getdate();

print $day_today[‘mday’];

print $day_today[‘wday’];

print $day_today[‘yday’];
```

So whichever piece of the array you need to get to goes between square sections. You then sort one of the keys between quote marks.

PHP AND MYSQL

PHP can associate with and control databases. The most prevalent database framework that is utilized with PHP is called Mysql. This is a free database framework, and accompanies the Wampserver programming you may have introduced toward the begin of the course. We will be working with Mysql databases all through these lessons.

Opening a Connection to a MySql Database

PHP has a considerable measure of inbuilt functions you can use to control databases. In PHP 5, a considerable measure more were included too! Here, we'll stay with the inbuilt functions for adaptations sooner than PHP 5. Anyhow, on the off chance that you have form 5, its well worth scrutinizing the fresher database capacities. A decent place to begin is php_page.net. To open our Customer Records database, we'll utilize the accompanying inbuilt capacities:

mysql_connect()

mysql_select_db()

mysql_close()

The approached we'll make has three strides. Here is an elaborated discussion on each of these steps.

Step 1 - Open an association with MySql

The main occupation is to connect with MySql. As its name recommends, mysql_connect( ) does precisely that.

Step 2 - Specify the database you need to open

In our code, we set up a variable with the name of our database:

$database_name = “customerrecords”;

Step 3 - Close the association

Shutting an association with a database is simple. In the event that you’ve utilized a record handle, you simply do this:

mysql_close($db_handle);

Perusing Records

To peruse records from a database, the method is as a rule to loop and find the ones you need. To tag which records you need, you utilize something many refer to as SQL. This stands for Structured Query Language. This is a regular, non-coding dialect that uses words like SELECT and WHERE. At its least difficult level, it’s genuinely direct. Be that as it may, the more unpredictable the database, the more trickier the SQL is. We’ll begin with something basic however. What we need to do, now that we have an association with our database, is to peruse all the records, and print them out to the page.

Structured Query Language

SQL is an approach to question and control databases. The fundamentals are not difficult to learn. On the off chance that you need to snatch the majority of the records from a table in a database, you utilize the word, SELECT. This can be done in the following manner:

SELECT * FROM Name_of_table

SQL is not case sensitive, so the above line could be composed as:

Select * From Name_of_table

However your SQL explanations are simpler to peruse on the off chance that you write the essential words in uppercase letters. The decisive words in the lines above are SELECT and FROM. The indicator (*) signifies “All Records”. The variable Name_of_table is the name of a table in your database. So the entire line peruses:

“SELECT all the records FROM the table called Name_of_table”

You don’t need to choose all the records from your database. You can simply select the sections that you require. For instance, in the event that we needed to choose simply the first name and surname segments from this table, we can determine that in our SQL String:

“SELECT f_name, s_name FROM tb_customer_records”;

At the point when this SQL proclamation is executed, just the f_name and s_name segments from the database will be returned. There are a considerable measure more SQL charges to get used to, and you’ll meet a greater amount of them as you come. For the time being, we’re simply selecting all the records from our table.

Inserting A Record

To add records to a table in your database, you utilize pretty much the same code as done a while ago. The main thing that needs to change is your SQL proclamation. The steps we’re going to be take are:

1. Open an association with MySql
2. Specify the database we need to open
3. Set up a SQL Statement that can be utilized to add records to the database table
4. Use mysql_query( ) once more, yet this time to add records to the table
5. Close the association

The syntax of the query is as follows:

INSERT INTO name_of_table (Column/s) VALUES (value/s for sections)

Anyhow, attempt to run your code now, and check whether its all meeting expectations appropriately. You ought to observe that you now have two records in your database table.

Creating A Table

You can make tables utilizing SQL (and entire databases), and indicate the fields you need to make in the table. In any case, doing it along these lines is not proposed. We have a tendency to overlook which fields are in the table, their information sorts, which field is the essential keys, and which ones are assigned to NULL. On the off chance that you can get a grasp of visual devices like phpmyadmin, then this can be much simpler for you to comprehend and do.

To make a table, you can utilize the CREATE word (known as a statement, in database terminology). Here's the SQL to make the basic location book we've been utilizing. This accepts that the database itself as of now exists, and that the PHP code to open an association has as of now been composed:

```
$my_sql="create TABLE Customerrecords

(

ID int(7) NOT NULL auto_increment,

f_name varchar(50) NOT NULL,

s_name varchar(50) NOT NULL,

email_id varchar(50),

PRIMARY KEY (ID),

UNIQUE id (ID)

)";

mysql_query($my_sql);
```

Updating A Record

You can likewise overhaul a record in your table. Of course, the word UPDATE is utilized for this. Here's a sample:

$my_sql = “UPDATE Customerrecords SET email_id = “email_address” WHERE f_name = ‘Jack’ and s_name = ‘Wright’”;

After the expression UPDATE, you require the name of the table you need to update. At that point you require an alternate keyword: SET. After the word SET, you write the name of the column you need to change. In the SQL above, we’re changing the email id. Anyway, recognize the WHERE provision. We’ve detailed that the record to change ought to have the f_name of Jack and the s_name of Wright.

Deleting A Record

On the off chance that you need to erase a record in a table, utilize the DELETE keyword. This can be performed in the following manner:

$my_sql = “DELETE FROM Customerrecords WHERE f_name = “Jack” AND s_name = ‘Wright’”;

After the DELETE word, you require FROM. At that point you write the name of the table. Next, you have to tag which record you need to erase. It’s a decent thought to verify your WHERE statement is going to be a unique value.

C Programming

The C Programming Language Guide For Beginners

Scott Sanderson

Table of Contents

BONUS BOOK OFFER 1

BONUS BOOK OFFER 2

BONUS BOOK OFFER 3

BONUS BOOK OFFER 4

C Language Overview

The C programming language is a universally used programming language that was initially created by Dennis M. Ritchie to create the UNIX working environment at Bell Labs. C was initially actualized on the DEC PDP-11 machine in the year 1972. However it was only in 1978 that Dennis Ritchie and Brian Kernighan created the first freely accessible version of C. This standard C version is commonly referred to as K&R standard.

The UNIX framework, the C compiler, and basically all UNIX applications and utilities have been composed in C. The C has now turned into a broadly utilized proficient programming language for many different reasons. These reasons have been mentioned below:

- Structured programming language
- Easy to learn
- Its ability to create efficient projects.
- Its ability to deal with low-level exercises.
- Its ability to incorporate a mixed bag of machine stages.

Factsheet

Before moving any further, let us look at a few facts about the C programming language.

- C is a successor of B programming language, which was presented around 1970.
- C was concocted to compose a working environment called UNIX.
- The UNIX OS was completely composed in C in the year 1973.

- The programming language was formalized in 1988 by ANSI (American National Standard Institute).
- Today's most popular operating system, Linux, and RBDMS, MySQL, have been composed in the C programming language.
- Most of the virtual applications have been actualized utilizing C.
- Today, C is the most generally utilized System Programming Language.

Why Use C?

C was at first utilized for framework improvement work, specifically for the projects that make up the working framework. C was received as a framework advancement programming language on the grounds that it delivers code that runs almost as quick as code written in low level computing constructs. A few samples of the utilization of C may be:

- Language Compilers
- Operating Systems
- Text Editors
- Assemblers
- Network Drivers
- Print Spoolers
- Databases
- Modern Programs
- Utilities
- Language Interpreters

C Programs

A C project can differ from 3 lines to many lines and it ought to be built into one or more source files with the '.c' extension. A few examples of c source files are sample.c and example.c. You can utilize "vi", "vim" or any other text editor to compose your C project. Text editor is a prerequisite for coding in the C programming language and has been illustrated in greater detail in the section to follow.

C Environment Setup

Text Editor

This section portrays how to set up your work environment before you begin doing your C programming. Before you begin, you require the accompanying two virtual applications accessible on your machine, (a) The C Compiler (b) Text Editor.

This will be utilized to prepare your system for the work you intend to do on it. Few of these editors include VIM, VI and Notepad on Windows. Name and version of the text editor may vary depending on the operating system and machine that you are using. For instance, Notepad will be utilized on Windows, and VI or vim can be utilized on windows and Linux or UNIX as well.

The source files you make with your text editor are called source files and contain source code, which in the present context shall be written in C programming language. The source files for C projects are named with the extension, '.c'. Before beginning your programming, verify you have one of the text editors installed on your system. Moreover, you must also have enough experience to compose using that machine, save it in a file, arrange it, and lastly execute it.

The C Compiler

The source code written in source file is the heart for your project. It needs to be compiled,

to transform this code into machine language code. The machine does not understand codes written in high-level languages. Therefore, transforming the code into its machine language equivalent allows the CPU to execute the project according to the guidelines given. This C programming language compiler will be utilized to translate your source code into an executable. Most regularly utilized and freely accessible C language compilers are GNU C/C++ compiler. Besides this, there are compilers by HP and Solaris as well that can be used on the off chance that you have the corresponding operating systems.

Your First C Program

Like with any other programming language, you cannot expect to learn a programming language unless you code in it. Going by the word and tradition, we introduce you to your first C program, which is a simple ‘Hello World!’ program. This program is chosen as your first face-to-face interaction with C because this program code is the absolute minimum C project structure. As you move forward, you will be exposed to more complex and intricate C codes.

Sample Code: sample.c

```
#include <stdio.h>

int main() {

    /* Sample Code for printing Hello World! As output */

    printf(“Hello World!”);

    return 0;

}
```

A C program essentially comprises of the accompanying parts:

- Comments
- Functions
- Statements & Expressions
- Variables
- Preprocessor Commands

Now that you have been introduced to the different aspects of a standard C program, let us take a closer look at the ‘Hello World!’ that we just wrote.

- Line 1:

The first line of the project #include <stdio.h> is a preprocessor directive, which advises a C compiler to include the contents of the header file, stdio.h, before compilation.

- Line 2:

The next line int main() is the principle function where program execution starts in C programming.

- Line 3:

The next line/*…*/ will be overlooked by the compiler and it has been put to comments or lines that are not an active part of the C code in the project. In other words, they are just remarks that are meant for the developer to refer to for understanding the code.

- Line 4:

The next line printf(…) is an alternate function accessible in C, which is responsible for the printing of the text:

“Hello World!”

- Line 5:

The next line give back 0, which is the value returned by the main () function upon completion to the system.

Compiling and Executing C Program

So, you have the code ready with you now. The next obvious steps include compiling and executing the code on the machine. The next few steps shall illustrate how the code is saved, compiled and run.

- Copy and paste the code into the text editor that you using for the project. If you using an integrated framework like Bloodshed C++ or Xcode for development, you can directly write the code in the text editor of these environments.
- Save the source file as sample.c or any other name that seems convenient to you, at the desired location in the system memory.
- Compile the code using console based commands or integrated shortcuts in the programming environment.
- If the file is compiled successfully, you will notice that sample.o and a.out files will be created at the same location as the source file.
- The a.out file can be used for executing the file.

Upon running the executable file, you shall notice the following expected output.

Expected Output:

Hi, World!

Verify that you are using the GCC compiler and running the file from the correct directory location.

Basic Syntax

This chapter will give insights about all the fundamental language structure of the C programming language. These structures entail keywords, tokens, identifiers, and so on. You have already seen the essential structure of C program. So, you already have a basic idea of what a C program contains and includes. In the sections to follow, we shall take a closer look at these structures and how they can be used in real-life programming.

Tokens in C

In order to illustrate the concept of tokens in C, let us take the following statement as example:

printf(“Print this text!”);

The individual tokens for this statement are:

- printf
- (
- “Hi, World! \n”
-) ;

A C system comprises of different tokens and a token can be an identifier, a keyword, a literal or a function call.

Semicolons (;)

Notice the use of semicolons at the end of each statement. Semicolon is the terminating sequence for C and it is mandatory to use semicolon at the end of each C statement. However, please note that semicolon should not be used at the end of preprocessor directives of #include statement.

Comments

C programming language allows the use of multi-line comments in the following format:

/* This is a comment */

Comments are basically help text in your C project. They are overlooked by the compiler and are not considered an active part of the project. Typically, they must begin with '/*' and end with the characters '*/' as shown in the example above. You can't have nested comments or comments in side comments. In other words, /* must have a matching */ and cannot be followed by another /* unless the first comment beginning sequence is terminated with a matching sequence.

Identifiers

A C identifier is a name used to distinguish a function, variable or any other construct created by the developer. An identifier begins with a letter or an underscore (_). However, the sequence to follow can contain alphabets, uppercase or lowercase, and digits (0 – 9). C does not permit accentuation characters, for example, @, $, and % as constituent characters in identifiers.

Please note that C language is case-sensitive. Therefore, the sequences, 'Character' and 'character' are different for the compiler. A few examples of identifier are:

- A_xyz
- a_xyz
- _inty
- a12b

Keywords

Some words are restricted for use by the developers as they are used by the language to

instruct the compiler on some basic functionality provided as part of the C programming language. Here is a list of the reserved keywords. You must have a keen look at these keywords and ensure that you don’t use identifier with names same as these keywords as this will result in an otherwise evitable compilation error.

- else
- auto
- switch
- long
- enum
- break
- typedef
- register
- extern
- case
- union
- return
- float
- char
- unsigned
- short
- const
- signed

- for
- continue
- void
- sizeof
- goto
- default
- volatile
- static
- if
- do
- while
- struct
- int
- double
- _packed

Whitespace in C

A line containing just whitespace, potentially with a comment in it, is known as a clear line, and a C compiler completely overlooks it. Whitespace is the term utilized as a part of C to portray tabs, spaces, comments and newline characters. Whitespace differentiates different parts of the statement from one another. Additionally, it empowers the compiler to distinguish where one component, for example, int, is closing and the following component is starting. For example, consider the following statement:

int a;

Here a whitespace separates ‘int’ from ‘a’.

There must be no less than one whitespace character (typically a space) between different components of a statement for the compiler to identify to them. However, as far as the use of operators, mathematical, logical or relational, is concerned, the use of space for differentiation is not mandatory; you may or may not use it.

Data Types

In the C programming language, data types allude to a framework utilized for pronouncing variables or functions of distinctive data types. The type of a variable decides the amount of space it will take and how the bit pattern saved in it is utilized. The fundamental classification used for data types is given below:

- Basic Types:

  They are number-crunching sorts and comprises of the two following types: (a) integer sorts and (b) floating point sorts.

- Enumerated sorts:

  They are again number sorts and are utilized to characterize variables that must be allocated discrete number values all through the system.

- Void type:

  The keyword void demonstrates that no value can be assigned.

- Derived sorts:

  They incorporate (a) Array, (b) Pointer, (c) Union, (d) Structure and (e) Function.

The array and structure data types are also referred to as aggregate types. The type of a function defines the kind of value the function will return upon termination. We will discuss the essential data types in the accompanying segments.

Integer Types

Here is a list of data types that follow under this category. In addition, the storage space occupied by them and their range are also specified for your reference.

- char

    - Allocated Memory: 1 byte
    - Range: -128 to 127 or 0 to 255
- signed char
    - Allocated Memory: 1 byte
    - Range: -128 to 127
- unsigned char
    - Allocated Memory: 1 byte
    - Range: 0 to 255
- int
    - Allocated Memory: 2 or 4 bytes
    - Range: -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
- short
    - Allocated Memory: 2 bytes
    - Range: -32,768 to 32,767
- unsigned short
    - Allocated Memory: 2 bytes
    - Range: 0 to 65,535
- unsigned int
    - Allocated Memory: 2 or 4 bytes
    - Range: 0 to 65,535 or 0 to 4,294,967,295
- long

    - Allocated Memory: 4 bytes
    - Range: -2,147,483,648 to 2,147,483,647
- unsigned long
    - Allocated Memory: 4 bytes
    - Range: 0 to 4,294,967,295

To get the precise size of a variable or data type, you can utilize the sizeof operator. The declarations sizeof(<data type>) yields the size of the data type or variable in bytes. Given below is an example, which illustrates the concept, discussed below:

```
#include <limits.h>

#include <stdio.h>

int main() {

printf("Data type char (size in bytes): %d \n", sizeof(char));

return 0;

}
```

Upon compilation and execution of this code, you must get the following output:

Data type char (size in bytes): 1

Floating Point Data Types

Here is a list of data types that follow under this category. In addition, the storage space occupied by them, their range and precision value are also specified for your reference.

- float
    - Allocated Memory: 4 byte
    - Range: 1.2e-38 to 3.4e+38

    - Precision: 6 decimal places
- double
    - Allocated Memory: 8 byte
    - Range: 2.3e-308 to 1.7e+308
    - Precision: 15 decimal places
- long double
    - Allocated Memory: 10 byte
    - Range: 3.4e-4932 to 1.1e+4932
    - Precision: 19 decimal places

The header file named float.h characterizes macros that permit you to utilize these data type values. The following code will allow you to find the exact amount of allocated memory in bytes on your system for the concerned data type.

```
#include <float.h>

#include <stdio.h>

int main(){

printf(“Allocated Memory for float : %d \n”, sizeof(float));

printf(“Precision: %d\n”, FLT_DIG );

printf(“Max Range Value: %E\n”, FLT_MAX );

printf(“Min Range Value: %E\n”, FLT_MIN );

return 0;

}
```

Upon compilation and execution of this code, you must get the following output:

Allocated Memory for float : 4

Precision: 6

Max Range Value: 3.402823E+38

Max Range Value: 1.175494E-38

The void Type

The void data type points out that no value is accessible. It is utilized as a part of three sorts of circumstances:

- Void returned by a function

  You must have commonly noticed the use of the data type void as return type of function. If not, you will see an extensive use of the same as you move forward in your experience with C. The void data type signifies that the function will not return anything. Example of such an implementation is:

  void print(int);

- Function Arguments as void

  There are different functions in C, which don’t acknowledge any parameter. A function with no parameter can acknowledge as a void. Example of such an implementation is:

  int print(void);

- Pointers to void

  A pointer of sort void * signifies the location of a variable. For instance, consider the following declaration:

  void *malloc(size_t size);

  This function returns a pointer to void. In other words, this function can return a

pointer to a location of any type.

You may not be able to comprehend the use and meaning of the void data type in entirety right now. However, as you move forward, you will find it easier to relate to and use this data type in your code.

Variables in C

A variable is only a name given to a region of the memory that our C projects can control. Every variable in C has a particular data type, which decides the size and format of the variable's memory, the scope of values that can be put inside that memory and the set of operations that can be performed to the variable.

The name of a variable can be made out of digits, letters and the underscore character. However, it must start with either a letter or an underscore. Uppercase and lowercase letters are different in light of the fact that C is case-sensitive. Here is the list of data types that are included in the C programming language.

- Char

  Char data type is a solitary octet (one byte).

- Int

  Int is the most regular size of a number for the machine.

- Float

  A solitary accuracy floating point value is represented by float.

- Double

  A double data type has double precision.

- Void

  Speaks to the nonattendance of any data type.

C programming language likewise permits to characterize different sorts of variables, which we will cover in consequent parts of the book like Pointer, Enumeration, Structure, Array and Union, in addition to some others. For this part, let us consider just essential

variable types.

Variable Definition in C

A variable definition intends to tell the compiler where and the amount of memory required to make for the variable. A variable definition defines the datatype of the variable and type of variable in terms of the number of individual variable elements that the variable shall contain. Typical variable declaration is given as:

<data-type> <names of the variables>;

Here, data-type must be a legitimate C data type. Therefore, it can be char, int, float or double. However, the names of variables can be any legitimate identifier names like i, I, _intx or _i12. These variables must be separated by commas and terminated with a semicolon. A few examples of such a statement include:

char ch, name;

int i, j;

float i, j;

double i, j;

As can be seen in the declarations given above, the first statement declares two variables ch and name of the char data type. Similarly, the other statements declare the variables i and j.

Now that you have declared the variable names and specified the data type that they shall be of, it is time to assign values to these variables. It is possible to assign values to variables in C using two ways. Value assignment can be done as part of the declaration or can be performed as a different statement.

int k = 0;

In this statement, the variable k is declared as an int and assigned the value 0.

```
int k;

k = 0;
```

In the set of statements given above, the variable k is declared as an int in the first statement. However, assigning of the value 0 is done in the second statement.

Examples of Variable Declaration in C:

```
int i = 0, j = 100;

static int i = 0, j = 100;

char z = ‘a’;
```

Sample Implementation:

Attempt the accompanying code on your machine. In this code sample, variables have been declared at the very beginning of the code. However, they have been characterized and introduced inside the main function:

```
#include <stdio.h>

int main ()

{

float fvar;

int cvar;

int avar, bvar;

avar = 0;

bvar = 1;

cvar = avar + bvar;
```

```
printf(“cvar = %d \n”, cvar);

fvar = 12.0/4.0;

printf(“fvar = %f \n”, fvar);

return 0;

}
```

Upon compilation and execution of this code, the following output shall be yielded.

```
cvar = 1

fvar = 3
```

Lvalues and Rvalues in C

There are two sorts of outflows in C:

- Lvalue

  An outflow that is lvalue may show up as either the left-hand or right-hand side of a task.

- Rvalue

  An outflow that is rvalue may show up on the right- yet not left-hand side of a task.

Variables are lvalues. Therefore, they may show up on the left-hand side of a task. Numeric literals are rvalues. Thus, they must show up on the left-hand side. As a result, the following assignment is legitimate in C.

```
int ivar = 20;
```

However, the following statement is incorrect and will generate a compilation error.

```
20 = ivar;
```

Constants and Literals

The constants allude to altered values that the system may not adjust during its execution. These variables are additionally called literals. Constants can be of any of the fundamental types like char, int or float. In addition, they may also be of the string literal or array type. The constants are dealt in much the same manner as customary variables aside from the fact that their values can’t be changed after their definition.

Integer Literals

An integer literal can be hexadecimal, octal, or decimal. A prefix defines the base or radix of the defined number.

- Hexadecimal: 0x or 0x
- Octal: 0
- Decimal: None

An integer literal can likewise have an addition that is a mix of U and L, for unsigned and long, individually. The addition can be uppercase or lowercase and can be in any form.

Here are a few legitimate cases of integer literals:

- 212
- 215u
- 0xfee

On the other hand, the following declarations are not allowed.

- 018

  Please note the use of 8 in the number. Octal numbers cannot contain 8 as their range is only from 0 to 7.

- 011uu

Please note the rehashing of the postfix. This is not allowed in C.

Floating Point Literals

Typically, a floating-point number includes a number part, a decimal point, a fractional part, and an exponentiation part. However, floating point numbers may exist in exponentiation or decimal structure. A few examples of these literals are given below for your understanding and reference.

- 3.14159 /* Legal */
- 314159e-5l /* Legal */

The following examples are not legitimate declarations.

- 510e
- .e55

Please note the missing decimal of exponential part in the declarations.

Character Constants

Character literals are encased in single quotes. For instance, ‘a’ is a character constant. However, this type of a literal can hold alphanumeric characters, special characters or general characters. There are sure characters in C when they are gone before by an oblique punctuation line. They will have unique importance and are utilized to mention special punctuations like tab (\t) or newline (\n). The following list illustrates the different escape sequences and their meanings.

- Character
    - \” ”
    - ' ”

- \ \
- \? ?

- Backspace - \b
- Alert or chime - \a
- Newline - \n
- Form Carriage - \f
- Horizontal tab - \t
- Carriage return - \r
- Hexadecimal number (1 or more digits) - \xhh …
- Octal number (1-3 digits) - \ooo
- Vertical tab - \v

A sample code that illustrates the use of these characters and their significance is given below.

```
#include <stdio.h>

int main () {

printf(“Hey \t Everyone \n\n”);

return 0;

}
```

Upon compilation and execution of this code, the following output can be expected

Hey Everyone

String Literals

String literals or constants are encased in double quotes ” “. A string contains characters that are like character literals: plain characters, escape characters and other characters. You can break a long line into various lines utilizing string literals and dividing them utilizing white spaces. Here are a few cases of string literals. All the three structures are indistinguishable strings.

- “hi, \
- “hi, dear”
- dear”

How to Declare Constants

There are two straightforward ways in C for declaration of constants:

1. Using const keyword
2. Using #define preprocessor

The #define Preprocessor

This structure can be utilized using the #define preprocessor. The following implementation illustrates the use of this method.

```
#include <stdio.h>

#define TAB “\t”

#define LEN 10

int main () {

int x;
```

```
x = LEN * 100;

printf(“x = %d”, x);

printf(“%c”, TAB);

printf(“x = %d”, x);

return 0;

}
```

Upon compilation and execution of the above code, the following result can be expected:

```
1000            1000
```

The const Keyword

You can utilize const prefix for declaration of constants with a particular data type:

```
const <data type> <variable name> = <value>;
```

Here is a code that illustrates the use of this concept:

```
#include <stdio.h>

int main ()

{

const int LEN = 10;

const char TAB = ‘\n’;

printf(“LEN = %d”, LEN);

printf(“%c”, TAB);

printf(“LEN = %d”, LEN);

return 0;
```

```
}
```

Upon compilation and execution of the above code, the following result can be expected:

```
1000        1000
```

Note the use of capital letters for naming the constant variables. Although, this is not a mandatory practice, it is certainly a good programming practice.

Storage Classes

A storage class defines the extension and lifetime of variables and/or functions inside a C Program. These keywords are usually placed before the data type of the variable or function. There are several C storage classes available in C, which are as follows:

- register
- auto
- static
- extern

The auto Storage Class

The auto class is the default storage class for all variables in C programming. It is used in the following manner in a program:

auto int x;

The illustration above declares a variable x, which is declared auto. This storage class is used for local variables of a function.

The register Storage Class

The register class is utilized for declaring local variables that ought to be put away in a register rather than RAM. This implies that the variable has a most extreme size equivalent to the register size (typically single word) and can’t have the unary “&” administrator connected to it as it doesn’t have a memory area. Typical implementation of a variable of this class is as follows:

```
register int  x;
```

The register ought to just be utilized for counters. It ought to likewise be noted that characterizing “register” does not imply that the variable will be put away in a register. It implies that the variable may be put away in a register if the environment and system allows.

The static Storage Class

The static class orders the compiler to keep a local variable alive throughout the execution of the program. Thusly, making local variables static permits them to keep up their values between calls of functions. The static modifier might likewise be associated to global variables as well. At the point when this is carried out, it causes that variable’s degree to be confined to the file in which it is declared. Sample implementation that utilizes this storage class variable is given below:

```
#include <stdio.h>

void samplefunc (void);

static int tally = 10;

int main () {
        while(tally- -)
        {
                samplefunc ();
        }
        return 0;
}

void samplefunc (void)
```

```
{
    static int lvar = 10;
    lvar++;
    printf("lvar is %d and tally is %d\n", lvar, tally);
}
```

You may not comprehend this illustration at this point on the grounds that I have utilized both local as well as global variables. So until further notice, let us continue regardless of the fact that you don't comprehend it totally. At the point when the above code is compiled and executed, the following output may be expected.

lvar is 11 and tally is 9

lvar is 12 and tally is 8

lvar is 13 and tally is 7

lvar is 14 and tally is 6

lvar is 15 and tally is 5

The extern Storage Class

The extern class is utilized to give a reference of a global variable that is noticeable and accessible to all the source files of the project. When you have various files and you declare a function or global variable, which will be utilized as a part of different files additionally, then extern is used to instruct the compiler that the reference is being made to the variable or function that already exists in another file of the project. The extern modifier is most regularly utilized when there are two or more files using the same global variable or function. The following code illustrates the concept and use of global variables.

Filename: main.c

```
#include <stdio.h>

int count ;

extern void ext_write ();

int main ()

{

        ext_write ();

}
```

Second File: write_e.c

```
#include <stdio.h>

extern int count;

void ext_write (void)

{

count = 5;

printf(“count = %d\n”, count);

}
```

Here, extern is consistently used to declare the variables and functions of one file into another file. Upon compilation and execution of the project, which includes the two files shown above, the following result can be expected to occur.

```
count = 5
```

Operators

The operator set in C is extremely rich. Broadly, the operators available in C are divided in the following categories.

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators

Operations in C are used in essentially the same manner as in algebra. They are used with variables for performing arithmetic operations. Here is a list of arithmetic operators available in C.

Operation	Operator	Description
Addition	+	Adds the values of two variables
Subtraction	-	Subtracts the values of two variables
Multiplication	*	Multiplies the values of two variables
Division	/	Divides the values of two variables
Modulus	%	The resultant value is the the remainder of division
Increment	++	Increases the value by 1
Decrement	—	Decreases the value by 1

The Relational Operators

C also supports several relational operators. The list of relational operators that are supported by C are given below.

Operation	Operator	Description
Equal To	==	Compares the values of two variables for equality
Not Equal To	!=	Compares the values of two variables for inequality
Greater Than	>	Checks if one value is greater than the other value
Lesser Than	<	Checks if one value is lesser than the other value
Greater Than Or Equal To	>=	Checks if one value is greater than or equal to the other value
Lesser Than Or Equal To	<=	Checks if one value is lesser than or equal to the other value

The Bitwise Operators

The bitwise operators available in C can be easily applied to a number of data types. These data types include byte, short, long, int and char. Typically, any bitwise operator performs the concerned operation bitwise. For instance, if you consider the example of an integer x, which has the value 60. Therefore, the binary equivalent of x is 00111100. Consider another variable y, with the value 13 or 00001101. If we perform the bitwise operation & on these two numbers, then you will get the following result:

x&y = 0000 1100

The table shown below shows a list of bitwise operators that are available in C.

Operation	Operator	Description
BINARY AND	&	Performs the AND operation
BINARY OR	|	Performs the OR operation
BINARY XOR	^	Performs the XOR operation
ONE'S COMPLEMENT	~	Performs the complementation operation on a unary variable
BINARY LEFT SHIFT	<<	Performs the left shifting of bits
BINARY RIGHT		Performs the right shifting of

SHIFT	>>	bits

In addition to the above mentioned, C also supports right shift zero fill operator (>>>), which fills the shifted bits on the right with zero.

The Logical Operators

Logical operators are an integral part of any operator set. The logical operators supported by C are listed in the table below.

Operation	Operator	Description
Logical AND	&&	Returns True if both the conditions mentioned are true
Logical OR	||	Returns True if one or both the conditions mentioned are true
Logical NOT	!	Returns True if the condition mentioned is False

The Assignment Operators

There are following assignment operators supported by C language:

Operation	Operator	Description
Simple assignment operator	=	Assigns a value on the right to the variable in the left
Add - assignment operator	+=	Adds the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left
Subtract - assignment operator	-=	Subtracts the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left
Multiply - assignment operator	*=	Multiplies the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left
Divide - assignment operator	/=	Divides the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left

Modulus - assignment operator	%=	It takes the modulus of the LHS and RHS and assigns the resultant to the variable on the left
Left shift - assignment operator	<<=	It takes the left shift of the LHS and RHS and assigns the resultant to the variable on the left
Right shift - assignment operator	>>=	It takes the right shift of the LHS and RHS and assigns the resultant to the variable on the left
Bitwise - assignment operator	&=	It takes the bitwise AND of the LHS and RHS and assigns the resultant to the variable on the left
bitwise exclusive OR - assignment operator	^=	It takes the bitwise XOR of the LHS and RHS and assigns the resultant to the variable on the left
bitwise inclusive OR - assignment	|=	It takes the bitwise OR of the LHS and RHS and assigns the

operator		resultant to the variable on the left

Misc Operators

In addition to the above mentioned, there are several other operators, which are supported by C.

Conditional Operator (? :):

The conditional operator is a ternary operator that contains three operands. Essentially, this operator is used for the evaluation of boolean expressions. The operator tests the first operand or condition and if the condition is true, then the second value is assigned to the variable. However, if the condition is false, the third operand is assigned to the variable. The syntax of this operator is as follows:

variable a = (<condition>) ? valueiftrue : valueiffalse

Sample implementation:

```
#incluse<stdio.h>

int main(){

int x, y;

x = 5;

y = (x == 5) ? 15: 40;

printf( “y = ” +  y );

y = (x == 34) ? 60: 95;

printf( “x = ” + y );

}

}
```

The compilation and execution of this code shall give the following result:

y = 15

y = 95

Sizeof() Operator

The sizeof() operator returns the size of the variable in bytes. This operator can be used in the following manner:

int x;

x = sizeof(int);

Pointer to Variable (*)

The asterisk sign when used before a variable indicates the pointer to a variable. This operator can be used in the following manner:

int *ptr NULL;

ptr is a pointer that points to NULL.

Address of Variable (&)

The ampersand sign when used before a variable gives the address of the concerned variable. This operator can be used in the following manner:

int x;

int *ptr = &x;

Precedence of C Operators

More often than not, operators are used in combinations in expressions. However, you must have also realized that it becomes difficult to predict the order in which operations will take place during execution. The operator precedence table for C shall help you predict operator operations in an expression deduction. For instance, if you are performing addition and multiplication in the same expression, then multiplication takes place prior to addition. The following table illustrates the order and hierarchy of operators in C. The associativity for all the operators is left to right. However, the unary, assignment and conditional operator follows right to left associativity.

Operator	**Category**
() [] . (dot operator)	Postfix
++ - - ! ~	Unary
* / %	Multiplicative
+ -	Additive
>> >>> <<	Shift
> >= < <=	Relational
== !=	Equality
&	Bitwise AND

|	Bitwise OR
^	Bitwise XOR
&&	Logical AND
||	Logical OR
?:	Conditional
= += -= *= /= %= >>= <<= &= ^= |=	Assignment
,	Comma

Loops in C

Looping is a common programming situation that you can expect to encounter rather regularly. Loop can simply be described as a situation in which you may need to execute the same block of code over and over. C supports three looping constructs, which are as follows:

- for Loop
- do…while Loop
- while Loop

In addition to this, the foreach looping construct also exists. However, this construct will be explained in the chapter on arrays.

The while Loop:

A while loop is a control structure that permits you to rehash an errand a specific number of times. The syntax for this construct is as follows:

while(boolean_expression) {

/Statements

}

At the point when executing, if the boolean_expression result is genuine, then the activities inside the circle will be executed. This will proceed till the time the result for the condition is genuine. Here, key purpose of the while loop is that the circle may not ever run. At the point when the interpretation is tried and the result is false, the body of the loop will be skipped and the first proclamation after the whole circle will be executed.

Sample:

#include<stdio.h>

int main(){

int i=5;

while(i<10) {

printf(" i = " + i);

i++;

printf("\n");

}

}

This would deliver the accompanying result:

i = 5

i = 6

i = 7

i = 8

i = 9

i = 5

The do…while Loop

A do…while loop is similar to the while looping construct aside from that a do…while circle is ensured to execute no less than one time. The syntax for this looping construct is as follows:

```
do {

/Statements

}while(<booleanexpression>);
```

Perceive that the Boolean declaration shows up toward the end of the circle, so the code execute once before the Boolean is tried. In the event that the Boolean declaration is genuine, the stream of control bounced go down to do, and the code execute once more. This methodology rehashes until the Boolean articulation is false.

Sample implementation:

```
#include<stdio.h>

int main(){

int i = 1;

do{

printf(“i = ” + i );

i++;

printf(“\n”);

}while( i<1 );

}
```

This would create the accompanying result:

i = 1

The for Loop

A for circle is a reiteration control structure that permits you to effectively compose a loop that needs to execute a particular number of times. A for looping construct is helpful when you know how often an errand is to be rehashed. The syntax for the looping construct is as follows:

The punctuation of a for circle is:

for(initialization; Boolean_expression; redesign)

{

/Statements

}

Here is the stream of control in a four circle:

- The introduction step is executed in the first place, and just once. This step permits you to pronounce and introduce any loop control variables. You are not needed to put an announcement here, the length of a semicolon shows up.
- Next, the Boolean outflow is assessed. In the event that it is genuine, the assemblage of the loop is executed. In the event that it is false, the assortment of the loop does not execute and stream of control hops to the following articulation past the for circle.
- After the group of the for circle executes, the stream of control bounced down to the overhaul explanation. This announcement permits you to overhaul any circle control variables. This announcement can be left clear, the length of a semicolon shows up after the Boolean declaration.
- The Boolean outflow is currently assessed once more. On the off chance that it

is genuine, the loop executes and the scope rehashes itself. After the Boolean declaration is false, the for loop ends.

Sample Implementation

```
#include<stdio.h>

int main(){

for(int i = 0; i < 5; i = i+1) {

printf(“i = ” + i );

printf(“\n”);

}

}
```

This would deliver the accompanying result:

```
i = 0

i = 1

i = 2

i = 3

i = 4
```

The break Keyword

The break keyword is utilized to stop the whole loop execution. The break word must be utilized inside any loop or a switch construct. The break keyword will stop the execution of the deepest circle and begin executing the following line of code after the ending curly bracket. The syntax for using this keyword is as follows:

```
break;
```

The Continue Keyword

The proceed with decisive word can be utilized as a part of any of the loop control structures. It causes the loop to quickly bounce to the following emphasis of the loop.

- In a four circle, the continue keyword reasons stream of control to quickly bounce to the overhaul articulation.
- In a while or do/while loop, stream of control instantly hops to the Boolean interpretation.

The syntax of using this keyword is as follows:

continue;

The Infinite Loop

If the condition is the loop is always true, the loop shall continue infinitely. Typically, the ‘for’ loop construct is used for this purpose. In order to implement the for loop, you don’t need to mention any of the three expressions that are written inside the for loop. Sample implementation of the infinite loop is given by:

```
#include<stdio.h>

int main(){

for(;;)

{

//statements inside infinite loop

.}
```

Since no conditional statement is present in for loop, the condition is assumed to be always true by the compiler. As a result, the loop becomes infinite. This loop can be terminated using an external interrupt (Ctrl + C).

Decision Making in C

There are two sorts of decision making constructs in C. They are:

- if constructs
- switch constructs

The if Statement:

An if constructs comprises of a Boolean outflow emulated by one or more proclamations. The syntax for using this construct is as follows:

if(<condition>) {

//Statements if the condition is true

}

In the event that the Boolean construct assesses to true, then the scope of code inside the if proclamation will be executed. If not the first set of code after the end of the if construct (after the end wavy prop) will be executed.

Sample Implementation:

#include<stdio.h>

int main(){

int i = 0;

if(i < 1){

printf(“The if construct is executing!”);

}

}

This would create the accompanying result:

The if construct is executing!

The if…else Statement

An if proclamation can be trailed by a non-compulsory else explanation, which executes when the Boolean outflow is false. The syntax for this construct is as follows:

if(<condition>){

//Executes if condition is true

}

else{

//Executes if condition is false

}

Sample Implementation:

#include<stdio.h>

int main(){

int i = 0;

if(i > 1){

printf(“The if construct is executing!”);

}

else{

printf(“The else construct is executing!”);

}

}

This would create the accompanying result:

The else construct is executing!

The if…else if Statement

An if proclamation can be trailed by a non-compulsory else if…else explanation, which is exceptionally helpful to test different conditions utilizing single if…else if articulation.

At the point when utilizing if , else if , else proclamations there are few focuses to remember.

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to numerous else if's and they must precede the else.
- If one of the if conditions yield a true, the other else ifs and else are ignored.

The syntax for using this decision making construct Is as follows:

```
if(condition_1){

//Execute if condition_1 is true

}

else if(condition_2){

//Execute if condition_2 is true

}

else if(condition_3){

//Execute if condition_3 is true

}

else
```

```
{

//Execute if all conditions are false

}
```

Sample Implementation:

```
#include<stdio.h>

int main(){

int i = 0;

if( i > 1 ){

printf(“The first if construct is executing!”);

}

else if(i == 0){

printf(“The second if construct is executing!”);

}

else{

printf(“The else construct is executing!”);

}

}
```

This would create the accompanying result:

The second if construct is executing!

Nested if…else Statement

It is legitimate to home if-else constructs, which implies you can utilize one if or else if

proclamation inside an alternate if or else if explanation. The syntax for using this construct Is as follows:

```
if(condition_1){
//Execute if condition_1 is true
        if(condition_2){
        //Execute if condition_2 is true
        }
}
else if(condition_3){
//Execute if condition_3 is true
}
else
{
//Execute if all conditions are false
}
```

Sample Implementation:

```
#include<stdio.h>
int main(){
int i = 1;
if( i >= 1 ){
printf("The if construct is executing!");
```

```
        if(i == 1){
        printf("The nested if construct is executing!");
        }
}
else{
printf("The else construct is executing!");
}
}
```

This would create the accompanying result:

The if construct is executing!

The nested if construct is executing!

The switch Statement

A switch construct permits a variable to be tried for equity against a rundown of values. Each one value is known as a case, and the variable being exchanged on is checked for each one case. The syntax for using this decision making construct is as follows:

```
switch(<condition>){
case value1:
//Statements
break;
case value2 :
//Statements
break;
```

default:

//Optional

}

The accompanying runs apply to a switch construct:

- The variable utilized as a part of a switch explanation must be a short, byte, char or int.
- You can have any number of case explanations inside a switch. Each one case is trailed by the value to be contrasted with and a colon.
- The value for a case must be the same type as the variable in the switch and it must be a steady or an exacting value.
- When the variable being exchanged on is equivalent to a case, the announcements after that case will execute until a break is arrived at.
- When a break is arrived at, the switch ends, and the stream of control bounces to the following line after the switch.
- Not each case needs to contain a break. In the event that no break shows up, the stream of control will fall through to consequent cases until a break is arrived at.
- A switch articulation can have a discretionary default case, which must show up toward the end of the switch. The default case can be utilized for performing an undertaking when none of the cases is true. No break is required in the default case. However, as per the convention, the use of the same is recommended.

Sample Implementation:

#include<stdio.h>

```
int main(){

char mygrade = ‘A’;

switch(mygrade)

{

case “A” :

printf(“Excellent Performance!”);

break;

case “B” :

printf(“Good Performance!”);

break;

default :

printf(“Failed”);

}
```

Aggregate and run above code utilizing different inputs to grade. This would create the accompanying result for the present value of mygrade:

Excellent Performance!

Nesting of switch case constructs are also allowed in C programming language. Therefore, you can have a switch construct inside a case construct.

Functions

A function is a gathering of statements that together perform a specified task. Each C project has at least one function called the main(). You can create partitions in your code and arrange them into independent functions. How you partition your code into diverse functions is dependent upon you, yet coherently the division normally is so each one function performs a particular task.

The declaration of a function informs the compiler about the name of the function, its return type and the parameters the function expects to get. However, it is the definition of the function that contains a collection of statements. The C standard library gives various inherent functions that your system can call. A function is known with different names like a subroutine or a technique, in addition to other names.

Definition of Function

The general manifestation of a function definition in C programming is as per the following structure:

return_type name_of_fn (list of parameters)

{

//statements inside the function

}

A function definition in C programming language comprises of a function header and a body. Here are all the parts of a capacity:

- Return Type:

  A function may or may not give back a value. The return_type is the datatype of the value the function returns. A few functions perform the sought operations

without giving back a worth. For this situation, the return_type, in this case, is void.

- Function Name:

This is the real name of the function. The name and the parameter list together constitute the signature of the function.

- Parameters

A parameter is similar to a placeholder. At the point when a function is conjured, you pass a value to the parameter. This value is alluded to as a parameter. The parameter rundown alludes to the sort and number of the parameters of a function. Parameters are non obligatory. Therefore, a function may not contain any parameters at all.

- Function Body:

The function body contains a gathering of articulations that characterize what the function does.

Sample Implementation

The following example illustrates how a function is declared and defined. The function concerned is max ( ), which takes two parameters, number_1 and number_2.

```
int max_num (int number_1, int number_2)
{
int f_result;
if (number_1 > number_2)
f_result = number_1;
else
```

```
f_result = number_2;

return f_result;

}
```

Declaration of Function

The declaration statement informs the compiler regarding a function name and how to call the function. Typically, a function is declared in the following manner:

```
return_type name_of_func ( list of parameters );
```

For the function that we defined in the previous section, the declaration can be one of the following:

```
int max_num (int number_1, int number_2);

int max_num (int, int);
```

Calling a Function

Parameter names are not vital in the declaration statement of a function. However, their data types must be explicitly mentioned. It is mandatory to declare a function before defining or using it. While making a C function, you must be clear about what the function needs to do. To utilize a function, you will need to call that function to perform the characterized undertaking.

At the point when a project calls a function, system control is exchanged to the called function concerned. A called function performs the characterized assignment, and when its return statement is executed or when the closing brace is encountered. To call a function, you essentially need to pass the obliged parameters alongside its name. Moreover, you must assign the value to a matching variable if the function is expected to return a value.

```
#include <stdio.h>
```

```
int max_num (int number_1, int number_2);

int main () {

int num_a = 100; int num_b = 200; int ret_val;

ret_val = max_num (num_a, num_b);

printf( “Max val = %d\n”, ret_val );

return 0;

}
```

The max_num () function’s definition is the same as the function defined previously.

The expected output of this program will be:

Max val = 200

Function Arguments

In the event that a function needs to utilize arguments, it must declare variables and the type of values that these variables will hold. These variables are known as the formal parameters of the function. The formal parameters act like other local variables inside the function.

While calling a function, there are two ways that arguments can be transferred from the calling function to the called function:

Function Call by Value

The call by value strategy for passing arguments to a function duplicates the real value of an argument into the formal parameter of the function. For this situation, changes made to the parameter inside the function have no impact on the argument itself. Naturally, C programming uses call by value system to pass arguments. The example illustrated above is a simple call by value.

Function Call by Reference

The call by reference strategy for passing arguments to a function duplicates the location of a contention into the formal parameter. Inside the function, the location is utilized to get to the genuine argument utilized as a part of the call. This implies that progressions made to the parameter influence the passed argument.

To pass the argument by reference, argument pointers are sent to the function much the same as any other value. So, appropriately you have to proclaim the function parameters as pointer sorts as in the accompanying function swap (), which exchanges the values of the two number variables indicated by the parameters to this function. The function illustrated below implements a function call by reference.

```
#include <stdio.h>

void swap_num (int *ptr_x, int *ptr_y);

int main () {

int num_a = 5; int num_b = 15;

printf(“Original Value of num_a = %d\n”, num_a );

printf(“Original Value of num_b = %d\n”, num_b );

swap_num (&num_a, &num_b);

printf(“Post-swapping value of num_a = %d\n”, num_a );

printf(“Post-swapping value of num_b = %d\n”, num_b );

return 0;

}
```

Upon compilation and execution, this code shall produce the following output:

```
Original Value of num_a = 5
```

Original Value of num_b = 15

Post-swapping value of num_a = 15

Post-swapping value of num_a = 5

It is evident from this discussion that the original parameter values are changed post-swapping. On the other hand, had the same functionality been implemented using pass by value, the output for this program had been:

Original Value of num_a = 5

Original Value of num_b = 15

Post-swapping value of num_a = 5

Post-swapping value of num_a = 15

Scope Rules

Scope is defined as an area within which a variable or function is active. Therefore, the variable or function is not accessible outside this region. The scope of any variable or function can be classified under three main categories and on the basis of this, variables are called -

- Local Variables

  Variables that are defined and declared inside a block or function are called local variables.

- Global Variables

  Variables that are defined and declared outside any block or function are called local variables.

- Formal Parameters

  Formal parameter is another name given to function parameter.

Arrays

C programming language allows the use of a data structure called an array. This array can store an altered size of successive components of the same data type. A structure called array is utilized to store a collection of data, yet it is regularly more helpful to think about a structure as an accumulation of variables of the same sort.

As opposed to declaring individual variables, for example, num_0, num_1, …, and num_99, you announce only one array variable. You can name the variable as num[100] and in such a case, num[0], num[1], … and num[99] to access individual elements or variables of the array. All arrays comprise of contiguous or linear memory areas. The 0^{th} element relates to the first component and the last location to the last component.

Defining Arrays

The syntax of declaring an array is as follows:

<data type> name_array [size_array];

double acc_bal [10];

The array declared in the above statement declares an array, which possesses a capacity to hold as many as 10 double variables. This is known as a 1-dimensional array. The size_array must be a number more than zero and data type can be any legitimate C data type.

How to Declare Arrays

You can introduce array in C either one by one or utilizing a solitary explanation in the following manner:

double bal [5] = {670.0, 267.5, 30.4, 137.0, 560.0};

Accessing Array Elements

Array elements can be accessed using the index values. For instance, if you wish to access the first element of the array, you can access it in the following manner:

double x;

x = bal [0];

A component is accessed by indexing the name of the array. This is carried out by putting the list of the component inside square brackets after the name of the array. Sample implementation

#include <stdio.h>

int main () {

int num [5];

/* num is an array with 10 elements of the data type int */

int x, y;

/* instate components of show n to 0 */

for (x = 0; x < 5; x++)

{

num[x] = x + 50; //This statement puts values into the array elements

}

for (y = 0; y < 5; j++)

{

printf(“num[%d] = %d\n”, y, num[y]);

}

return 0;

}

At the point when the above code is compiled and executed, it creates the accompanying result:

num[0] = 50

num [1] = 51

num [2] = 52

num [3] = 53

num [4] = 54

num [5] = 55

Multi-dimensional Arrays

Multi-dimensional arrays can be declared using the following general syntax.

<data type> name_array [arr_size_1][arr_size_2]…[arr_size_n];

C programming language permits multidimensional arrays. Here is the sample of a multidimensional array declaration:

int arr_3d[2][3][2];

Passing Arrays as Function Arguments

Arrays can be passed into functions as arguments using three ways. The first of these ways is to pass the pointer to the array. On the off chance that you need to pass an array as an argument, you would need to declare function formal parameter in one of emulating three ways and each of the three statement produce comparable results in light of the fact that each one tells the compiler that a number pointer is expected. Moreover, you can pass multidimensional array in the same manner as well.

```
void s_func(int *s_param)
{
.
.
.
}
```

The other way of passing an array is to send the whole array to the function. Sample implementation of this way is given below:

```
void s_func (int s_param[5])
{
.
.
.
}
```

The last way of passing an array into a function is described in the sample code given below:

```
void s_func (int s_param[])
{
.
.
.
}
```

Getting an Array from Function

As should be obvious, the length of the array doesn't make a difference in light of the fact that C performs no limits checking for the formal parameters. C programming does not permit returning of the whole array. Therefore, all that you can do is return the pointer to an array. In other words, in the event that you need to give back an array from a function, you would need to pronounce a function giving back a pointer as in the accompanying sample:

```
int * s_func()

{

.

.

.

}
```

Pointer to an Array

It is doubtless that you would not comprehend this part until you are through the section identified with Pointers in C. So, assuming that you have some knowledge about pointers, let us start our discussion on pointer to arrays.

To begin with, it is important to mention that the name of the array is the pointer to the beginning of the array. Therefore, you can use the following set of statements:

```
double *ptr;

double bal [5];

ptr = bal;
```

It is legitimate to utilize array names as pointers, and the other way around. Along these

lines, *(bal + 1) is an authentic method for getting to the information at bal [1]. When you store the location of first component in ptr, you can get to array components utilizing *ptr, *(ptr+1), *(ptr+2) and so on. The following code is the illustration to demonstrate all the ideas talked about above:

```
#include <stdio.h>

int main () {

double bal [3] = {45.0, 1.6, 5.2};

double *ptr;

int x;

ptr = bal;

printf( “Pointers being used - \n”);

for ( x = 0; k < 3; k++ )

{

printf(“*(ptr + %d) : %f\n”,  x, *(ptr + x) );

}

printf( “Offset being used - \n”);

for ( x = 0; x < 3; x++ )

{

printf(“*(bal + %d) : %f\n”,  x, *(bal + x) );

}

return 0;

}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

Pointer being used –

*(ptr + 0) : 45.000000

*(ptr + 1) : 1.600000

*(ptr + 2) : 5.200000

Offset being used –

*(bal + 0) : 45.000000

*(bal + 1) : 1.600000

*(bal + 2) : 5.200000

In the above illustration, ptr is a pointer to double, which implies it can store the address of a location that is capable of storing double data type information. Therefore *ptr will give the contents stored at the address denoted by ptr.

Pointers

C pointers are usually the most dreaded topic in C programming. However, let us start by saying that pointer is perhaps the easiest concept in C programming and is capable of fulfilling several programming needs of projects. One of these tasks is dynamic memory management.

As mentioned previously, every single variable is saved at a memory location. The variable contents can, not only be accessed using the name of the variable, but they can also be accessed using the address of the variable. The pointer holds the variable's address. Simply, the pointer is a variable that contains the address of the variable.

Please note that the ampersand operator can be used to access the address of the variable in the following manner:

```
int i;

int *ptr;

ptr = &I;
```

A sample implementation is given below to illustrate the concept in a clearer manner.

```
#include <stdio.h>

int main () {

int  variable_1;

char variable_2 [5];

printf("Variable_1 variable's address: %x\n", &variable_1  );

printf("Variable_2 variable's address: %x\n", &variable_2  );

return 0;
```

}

Upon compilation and execution of the code, the following result can be expected to appear on the output screen:

Variable_1 variable’s address: bff5a400

Variable_2 variable’s address: bff5a3f6

So, now you have an idea about the variable’s storage and how an address can be used to access the variable’s value.

What Are Pointers?

As mentioned previously, the pointer is any variable that stores the address of another variable. A pointer is typically declared in the following manner:

<data type> * <name of variable>;

in the declaration shown above, data type is any legitimate type of data allowed in C and the name of the variable is the identifier assigned to the pointer. Note the use of the asterisk operator for declaring a pointer. However, it is also noteworthy that this operator is used for multiplication as well. Here are sample declarations that are used for declaring pointers of different types:

int *ptr; //pointer to an int

char *ptr; //pointer to an char

float *ptr; //pointer to an float

double *ptr; //pointer to an double

Using Pointers

Pointers are used for several tasks or in some cases, to simplify the method of doing certain tasks. The typical steps that are used for this purpose include the following:

- Define a pointer
- Assign a legitimate address to the pointer variable
- Access the address and its contents using the pointer variable

Have a look at the following implementation for a better understanding of the concept:

```
#include <stdio.h>

int main () {

int  my_var = 405;

int  *my_ptr;

my_ptr = &my_var;

printf("Variable's address: %x\n", &my_var  );

printf("Value of address stored in the pointer: %x\n", my_ptr );

printf("Content of pointer: %d\n", *my_ptr );

return 0;

}
```

Upon compilation and execution of the above code, the following output can be expected to appear:

```
Variable's address: bff5a400

Value of address stored in the pointer: bff5a400

Value of address stored in the pointer: 405
```

NULL Pointers in C

You must have noticed in the above code that pointer declarations do not contain an assignment. In order to ensure that there is no scope of error because of pointer declaration

and assignment, it is considered a good programming practice to assign NULL to pointers. Any pointer that has been assigned to NULL is referred to as NULL pointer. In most work environments, the value of NULL is defined in standard libraries as 0. Sample implementation for this concept is given below:

```
#include <stdio.h>

int main () {

int  *my_ptr = NULL;

printf("my_ptr's NULL value = : %x\n", &ptr  );

return 0;

}
```

Upon compilation and execution of the code, the following output shall appear:

my_ptr's NULL value = 0

On the majority of the OS, projects are not allowed to get to memory at address 0 on the grounds that the OS, for its own use, saves that memory. Then again, the memory address 0 has exceptional centrality; it flags that the pointer is not planned to indicate an open memory area. However by tradition, if a pointer contains the invalid (zero) value, it is accepted to indicate nothing.

To check for an invalid pointer you can utilize an if construct as shown in the code excerpt below:

```
if(!ptr) //indicates the pointer is invalid

if(ptr) //indicates the pointer is valid
```

Operations on Pointers

As clarified already, C pointer is a location, which contains a numeric value. Accordingly,

you can perform operations on a pointer just as you manipulate any other numeric variable. There are four math operators that can be utilized on pointers: ++, -, +, and -. To comprehend pointer operations, let us consider that ptr is an int pointer, which indicates the location 5000. Expecting 32-bit numbers, let us perform the accompanying math operation on the pointer:

Incrementing Pointer

If we perform the increment operation on a pointer using the following statement, you can expect the final value of the pointer to become 5004.

ptr++ ;

This operation will move the pointer to next memory location without affecting value of the memory area. In the event that ptr is a pointer to a character whose location is 5000, then above operation will make the ptr point to the location 5001 in light of the fact that next character will be accessible at 5001.

The use of pointers is usually preferred over arrays in light of the fact that arrays have fixed capacity and any runtime changes made to the data cannot be adjusted into the system. The following code describes the use of this concept:

```
#include <stdio.h>

const int MAXVAL = 3;

int main () {

int my_var[] = {1, 10, 20};

int x = 0;

int *my_ptr = NULL;

ptr = my_var;
```

```
for ( x = 0; x < MAXVAL; x++)

{

printf("my_var[%d] address = %x\n", x, my_ptr );

printf("my_var[%d] value = %d\n", x, *my_ptr );

my_ptr++;

}

return 0;

}
```

At the point when the above code is compiled and executed, it creates result as shown below:

```
my_var [0] address = bf882b30

my_var [0] value = 1

my_var [1] address = bf882b34

my_var [1] value = 10

my_var [2] address = bf882b38

my_var [2] value = 20
```

Decrementing Pointer

The same contemplations apply to decrementing a pointer, which declines its value by the quantity of bytes of its data type as appeared:

```
#include <stdio.h>

const int MAXVAL = 3;

int main () {
```

```
int my_var[] = {1, 10, 20};

int x = 0;

int *my_ptr = NULL;

ptr = &my_var[MAXVAL-1];

for ( x = MAXVAL; x >= 0 ; x—)

{

printf(“my_var[%d] address = %x\n”, x, my_ptr );

printf(“my_var[%d] value = %d\n”, x, *my_ptr );

my_ptr—;

}

return 0;

}
```

At the point when the above code is compiled and executed, it creates result as shown below:

```
my_var [2] address = bf882b38

my_var [2] value = 20

my_var [1] address = bf882b34

my_var [1] value = 10

my_var [0] address = bf882b30

my_var [0] value = 1
```

Pointer Comparisons

Pointers may be analyzed by utilizing relational operators like == (equal to), < (less than), and > (greater than). In the event that ptr_1 and ptr_2 point to variables that are identified with one another, for example, components of the same array, then ptr_1 and ptr_2 can be genuinely analyzed.

Sample implementation:

```
#include <stdio.h>

const int MAXVAL = 3;

int main () {

int my_var[] = {1, 15, 27};

int x;

int *my_ptr;

my_ptr = my_var;

x = 0;

while ( my_ptr <= &my_var[MAXVAL - 1] )

{

printf(“my_var[%d] address = %x\n”, x, my_ptr );

printf(“my_var[%d] value = %d\n”, x, *my_ptr );

my_ptr++;

x++;

}

return 0;

}
```

At the point when the above code is compiled and executed, it delivers the following result:

my_var [0] address = bf882b30

my_var [0] value = 1

my_var [1] address = bf882b34

my_var [1] value = 15

my_var [2] address = bf882b38

my_var [2] value = 27

Array of Pointers

Before we comprehend the idea of array of pointers, let us consider the accompanying case, which makes utilization of an array of 3 numbers:

```
#include <stdio.h>

const int MAXVAL = 3;

int main ()

{

int  my_var[] = {1, 15, 27};

int x;

for (x = 0; x < MAXVAL; x++)

{

printf("my_var[%d] = %d\n", x, my_var[x] );

}

return 0;
```

```
}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

```
my_var [0] value = 1

my_var [1] value = 15

my_var [2] value = 27
```

There may be a circumstance when we need to keep an array, which can store pointers to an int or char or some other data type. Pointer of arrays is declared in the following manner:

```
int *arr_ptr[MAXVAL];
```

This pronounces arr_ptr as an array of MAXVAL int pointers. In this manner, every component in arr_ptr, now holds a pointer to an int. Accompanying code makes utilization of three numbers, which will be stored in an array of pointers as shown below:

```
#include <stdio.h>

const int MAXVAL = 3;

int main () {

int my_var[] = {1, 15, 27};

int x;

int *arr_ptr[MAXVAL];

for ( x = 0; x < MAXVAL; x++)

{

arr_ptr[x] = &my_var[x];
```

```
}
for ( x = 0; x < MAXVAL; x++)
{
printf("my_var[%d] = %d\n", x, *arr_ptr[x] );
}
return 0;
}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

```
my_var [0] value = 1
my_var [1] value = 15
my_var [2] value = 27
```

Pointer to Pointer

A pointer to a pointer is a manifestation of various redirections, or a chain of pointers. Typically, a pointer contains the location of a variable. When we characterize a pointer to a pointer, the first pointer contains the location of the second pointer, which indicates the area that contains the real value as appeared. A variable that is a pointer to a pointer must be declared by putting an extra asterisk (*) before its name. Sample implementation:

```
int **my_double_ptr;
```

At the point when a target value is indicated by a pointer to a pointer, getting to that value requires the reference operator to be connected twice, as is demonstrated below in the illustration:

```
#include <stdio.h>
```

```
int main () {

int  my_var;

int  *my_ptr;

int  **my_pptr;

my_var = 2435;

my_ptr = &my_var;

my_pptr = &my_ptr;

printf(“my_var = %d\n”, my_var );

printf(“*my_ptr = %d\n”, *my_ptr );

printf(“**my_pptr = %d\n”, **my_pptr);

return 0;

}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

```
my_var = 2435

*my_ptr = 2435

**my_pptr = 2435
```

Passing Pointers to Functions

C programming language permits you to pass a pointer to a function. This concept is illustrated using the following code:

```
#include <time.h>

#include <stdio.h>
```

```
void get_secs(unsigned long *my_par);

int main () {

unsigned long my_sec;

get_secs( &my_sec );

printf("my_sec = %ld\n", sec );

return 0;

}

void get_secs(unsigned long *my_par)

{

*my_par = time( NULL );

return;

}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

```
my_sec = 12456668
```

Please note that any function that can accept a pointer can accept an array of pointers as well.

```
#include <stdio.h>

double get_avg (int *my_arr, int my_arr_size);

int main () {

int bal[3] = {50, 4, 345};

double my_avg;
```

```
my_avg = get_avg ( bal, 3 ) ;

printf(“Average = %f\n”, my_avg );

return 0;

}

double get_avg(int *my_arr, int my_arr_size)

{

int x, my_sum=0;

double my_avg;

for (x = 0; x < my_arr_size; ++x)

{

my_sum += my_arr[x];

}

my_avg = (double)my_sum/my_arr_size;

return my_avg;

}
```

At the point when the above code is compiled and executed, it delivers the accompanying result:

```
Average = 133.000000
```

Returning Pointer from Function

Just as a pointer can be sent into a function as argument, it can likewise be returned by the function. A declaration for such a construct can be done in the following manner:

```
int * sample_func ()
```

{

…

}

Strings

The string in C programming language is a one-dimensional exhibit of characters, which is ended by an invalid character called null, ‘\0’. In this way, a string contains the characters that are part of the string followed by a null. The string declaration shown below creates a string containing the characters ‘name’. Please note the extra character that has been used in the declaration. For instance, the word ‘name’ has 4 characters. The use of an array of length 5 includes the null string ending character.

char str_name[5] = {‘n’, ‘a’, ‘m’, ‘e’, ‘\0’};

char str_name[] = “name”;

You don’t put the invalid character toward the end of a string, in the second declaration. The C compiler naturally puts the “\0” toward the end of the string when it introduces the array. Here is a code sample that uses this concept:

#include <stdio.h>

int main () {

char str_name[5] = {‘n’, ‘a’, ‘m’, ‘e’, ‘\0’};

printf(“Message: %s\n”, str_name);

return 0;

}

At the point when the above code is compiled and executed, it creates the following result:

Message: name

String Functions

- strcat(str_1, str_2);

This function attaches the string str_2 towards the end of string str_1.

- strcpy(str_1, str_2);

This function duplicates string str_2 into string str_1.

- strcmp(str_1, str_2);

This function returns the value 0 if the two strings being compared are equal. Any value other than 0, if returned, indicates inequality of strings.

- strlen(str_1);

This function returns the length of the string concerned.

- strstr(str_1, str_2);

This function matches the two strings and returns the index of the first location where str_2 occurs in str_1.

- strchr(str_1, ch);

This function returns a pointer to the first occurrence of ch in the string str_1.

You can discover a complete rundown of C string related capacities in C Standard Library.

Structures

Structures in C permit you to collect different kind of variables that can hold data of different type together. The construct hence formed is referred to as a structure. Structures are utilized to create records like book inventory or employee information. If you consider the example of book inventory, you may need to track the accompanying characteristics about each one book:

- Book ID
- Author
- Title
- Subject

How To Declare Structures

To declare a structure, you must utilize the struct construct. The struct articulation includes different variables into a structure, which is now known by a name. The declaration of the structure can be done in the following manner:

struct [structure tagname]

{

//declare elements of the structure

} [one or more structure variable names];

The structure tagname is non compulsory and every element's definition is an ordinary variable definition. For example, int x; or char ch; or whatever other variable definition that may be required by your project can be declared inside this structure. Towards the end of the structure's definition, before the last semicolon, you can tag one or more structure

variables yet it is non compulsory. Here is the way you would declare the BookRecord structure:

```
struct BookRecord

{

int  book_id;

char  author[50];

char  title[50];

char  subject[100];

} my_book;
```

Accessing Members of Structures

To get to any part of a structure, we utilize the dot (.) operator. The access operator is coded as a period between the structure variable name and the structure element that you wish to access. You would utilize struct keyword for declaring variables of structure sort. Here is a sample code to help you understand this concept:

```
#include <string.h>

#include <stdio.h>

struct BookRecord

{

char  title[50];

char  author[50];

char  subject[100];

int  bookID;
```

```
};
int main( ) {
struct BookRecord myBook_1;
struct BookRecord myBook_2;
//Adding elements to the structure
strcpy( myBook_1.author, “Humpty”);
strcpy( myBook_1.title, “Humpty Dumpty”);
myBook_1.bookID = 10002976;
strcpy( myBook_1.subject, “Nursery Rhyme”);
//Printing the structure elements
printf( ” ID of Book: %d\n”, myBook_1.bookID);
printf( ” Title of Book: %s\n”, myBook_1.title);
printf( ” Author of Book: %s\n”, myBook_1.author);
printf( ” Subject of Book: %s\n”, myBook_1.subject);
return 0;
}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

```
ID of Book: 10002976
Title of Book: Humpty Dumpty
Author of Book: Humpty
```

Subject of Book: Nursery Rhyme

Structures as Function Arguments

You can pass a structure as an argument to a function in the same manner as you pass other variable or pointer. You would access structure variables in the same manner as you have gotten to in the above sample:

```
#include <string.h>

#include <stdio.h>

struct BookRecord

{

char  title[50];

char  author[50];

char  subject[100];

int  bookID;

};

void printBookRecord ( struct BookRecord mybook );

int main ( )

{

struct BookRecord myBook_1;

struct BookRecord myBook_2;

//Adding elements to the structure

strcpy( myBook_1.author, “Humpty”);

strcpy( myBook_1.title, “Humpty Dumpty”);
```

```
myBook_1.bookID = 10002976;

strcpy( myBook_1.subject, “Nursery Rhyme”);

//Printing the structure elements

printBookRecord ( myBook_1 );

return 0;

}

void printBookRecord ( struct BookRecord myBook )

{

printf( ” ID of Book: %d\n”, myBook.bookID);

printf( ” Title of Book: %s\n”, myBook.title);

printf( ” Author of Book: %s\n”, myBook.author);

printf( ” Subject of Book: %s\n”, myBook.subject);

}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

```
ID of Book: 10002976

Title of Book: Humpty Dumpty

Author of Book: Humpty

Subject of Book: Nursery Rhyme
```

Pointers to Structures

You can declare pointers to structures in very much alike way as you declare pointer to some other variable. The following construct can be used for performing this task:

struct BookRecord *s_ptr;

In order to attach a pointer to a structure, the following statement must be utilized:

S_ptr = &mybook_1;

The members of the structure can be accessed using the redirection operator (->) in the following manner:

s_ptr->title;

The above code can be restructured into the following arrangement to adjust the use of pointers to structures.

```
#include <string.h>
#include <stdio.h>
struct BookRecord
{
char  title[50];
char  author[50];
char  subject[100];
int  bookID;
};
void printBookRecord ( struct BookRecord *mybook );
int main ( )
{
struct BookRecord myBook_1;
struct BookRecord myBook_2;
```

```
//Adding elements to the structure
strcpy( myBook_1.author, “Humpty”);
strcpy( myBook_1.title, “Humpty Dumpty”);
myBook_1.bookID = 10002976;
strcpy( myBook_1.subject, “Nursery Rhyme”);
//Printing the structure elements
printBookRecord ( &myBook_1 );
return 0;
}
void printBookRecord ( struct BookRecord *myBook )
{
printf( ” ID of Book: %d\n”, myBook->bookID);
printf( ” Title of Book: %s\n”, myBook->title);
printf( ” Author of Book: %s\n”, myBook->author);
printf( ” Subject of Book: %s\n”, myBook->subject);
}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

```
ID of Book: 10002976
Title of Book: Humpty Dumpty
Author of Book: Humpty
```

Subject of Book: Nursery Rhyme

Unions

A union is a unique kind of data type accessible in C that empowers you to store distinctive data types in the same memory area. You can declare a union with numerous elements, yet one and only element of the union can contain a value at a time. Unions give a proficient method for utilizing the same memory area for multiple reasons. It is critical to mention here that the memory assigned to the union is equal to the size of the largest element in the union.

How to Declare Unions

To declare a union, you must utilize the union declaration, which is similar in construct to that for a structure. The union construct creates a new data type, which constitutes several other data types. The standard syntax used for declaring a union is as follows:

union [union tagname]

{

//Declaration of elements to be included in the union

} [union variablenames];

The union tagname is non-compulsory and every declaration included in the union construct is a typical variable declaration. Therefore, you can use any variable, array or pointer declarations. Toward the end of the union’s declaration, before the last semicolon, you can indicate one or more union variables. However, it is non-compulsory. The following code excerpt illustrates how a union can be declared and defined.

union myData {

char name[20];

int count;

```
float averageval;

} g_data;
```

Presently, a variable of type myData is created using the declaration. This datatype can store a number, a floating-point number, or a series of characters. The memory possessed by a union will be sufficiently huge to hold the biggest component of the union. For instance, in above sample myData will possess 20 bytes of memory space in light of the fact that this is the greatest space, which will be used by character string. The following code shows how you can compute the memory size occupied by a union.

```
#include <string.h>

#include <stdio.h>

union myData {

char name[20];

int count;

float averageval;

} ;

int main( ) {

union myData data_1;

printf( “Memory of data_1 = %d\n”, sizeof(data_1));

return 0;

}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

Memory of data_1 = 20

Accessing Union Members

To get to any component of a union, we utilize the dot operator (.). This operator is coded as a period between the union variable name and the union component that you wish to access. You would utilize union keyword to declare variables of union sort. Sample implementation for this concept is given below:

```
#include <string.h>

#include <stdio.h>

union myData {

char name[20];

int count;

float averageval;

} ;

int main( ) {

union myData data_1;

strcpy( data_1.name, “Coding”);

data_1.count = 100;

data_1.averageval = 22.45;

printf( “data_1.name = %s\n”, data_1.name);

printf( “data_1.count = %d\n”, data_1.count);

printf( “data_1.averageval = %f\n”, data_1.averageval);

return 0;
```

```
}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

```
data_1.name = abfjkehf98

data_1.count = 3628648247

data_1.averageval = 22.45;
```

It can be seen in the output of the code that the value of name and count are garbage or are lost for some reason. The reason for this mishap is that the last assignment is done for averageval. Therefore, this is the only value that persisted as only one of the elements of the union had a value at a time. In order to ensure that each of the three values are printed correctly, the program should be converted to:

```
#include <string.h>

#include <stdio.h>

union myData {

char name[20];

int count;

float averageval;

} ;

int main( ) {

union myData data_1;

strcpy( data_1.name, “Coding”);

printf( “data_1.name = %s\n”, data_1.name);
```

```
data_1.count = 100;

printf( “data_1.count = %d\n”, data_1.count);

data_1.averageval = 22.45;

printf( “data_1.averageval = %f\n”, data_1.averageval);

return 0;

}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

```
data_1.name = Coding

data_1.count = 100

data_1.averageval = 22.45;
```

Note that the values are printed correctly in this attempt.

Header Files

Header file is a c file, which is saved with the extension .h and contains macro definitions and C function declarations. In addition, several c files may also be included in the header file. There are two sorts of header files: the files that the developer composes and the files that are part of the C compiler package.

You ask for the utilization of a header file in your project by including it, with the C preprocessing directive #include like you have seen the inclusion of stdio.h header file in most of the code that we have written in this book.

Including a header file is equivalent to replicating the content of the header file yet we don’t do it in light of the fact that it will be all that much error prone and it is not a decent thought to duplicate the header file in the source files on the off chance that we have different source files required by the project.

A basic practice in C or C++ projects is that we keep all the constants, macros, global variables, and declaration statement for functions in header files and include that header file wherever it is needed.

Include Operation

In order to include a header file, you must use one of the statements given below:

#include <filename>

#include ” filename ”

This structure is utilized for framework header records. It looks for a record named document in a standard rundown of framework indexes. You can prepend registries to this rundown with the -I alternative while arranging your source code.

This structure is utilized for header files of your own project. It looks for a file named

filename in the index or directory location containing the current file. You can prepend indexes to this rundown with the -I choice while compiling your source code.

char *s_test (void);

The #include operation lives up to expectations by coordinating the C preprocessor to output the detailed file as information before proceeding with whatever is left of the current source file contents. The yield from the preprocessor contains the yield officially created, followed by the yield coming from the included document, emulated by the yield that originates from the content after the #include directive.

Once-Only Headers

On the off chance that a header file happens to be incorporated twice, the compiler will process its contents twice. As a result, same variable and function declaration may be repeated. This shall inevitably lead to a compilation error. In order to avoid this situation, it is best to use the following construct:

#ifndef H_file

#define H_file

//Header files to be included

#endif

This construct is usually known as a wrapper #ifndef. At the point when the header is incorporated once more, the #ifndef directive returns a false and the #include statements are not processed. The preprocessor will skirt the whole content of the files concerned, and the compiler won’t see it twice.

Computed Includes

Now and then, it is important to choose one of a few diverse header documents to be incorporated into your project. They may declare setup parameters to be utilized on

diverse sorts of working frameworks. You could do this with an arrangement of conditionals like the ones given below:

```
#if Sys_1

# include “sys_1.h”

#elif Sys_2

# include “sys_2.h”

#elif Sys_3

…

#endif
```

At the same time, you must notice that the declaration becomes repetitive. Therefore, the preprocessor offers to utilize a macro for the header name. This is known as a computed include. As opposed to composing a header name as the immediate argument of #include, you basically put a macro name there:

```
#define Sys_h “sys_1.h”

…

#include Sys_h
```

Sys_h will be computed, and the preprocessor will search for sys_1.h as though the #include had been composed the way it was initially done. Sys_h could be declared in the makefile using the -D compiler option.

Typecasting

Typecasting is an approach to change over a variable of one data type into another data type. Typically, typecasting is done using the following construct:

(<data_type>) declaration

For instance, in the event that you need to store a long value into a straightforward integer number then you can type cast long to int. You can change over values starting with one sort then onto the next unequivocally utilizing the give utility. Consider the accompanying sample where the cast operator causes the division of one integer variable by an alternate to be executed as a floating-point number.

```
#include <stdio.h>

int main ()

{

int x = 21, y = 4;

double div;

div = (double) x/y;

printf(“Value of div = %f\n”, div );

}
```

Upon compilation and execution, the following output shall be expected.

Value of div = 5.250000

It ought to be noted here that the cast operator has priority over division, so the estimation of div is initially changed over to data type double. As a result, the value obtained after the division is double and can be assigned to div, which is also a double type of variable.

Typecasting can be implied as well. The compiler performs this form of typecasting or it can be done forcefully through the utilization of the cast operator. It is viewed as good programming practice to utilize the cast operator at whatever point type casting is desired.

Integer Promotion

The Integer promotion is the procedure by which estimations of number type “littler” than int or unsigned int are changed over either to int or unsigned int. Consider a case of including a character in an int:

```
#include <stdio.h>

int main () {

int  k = 5;

char ch = ‘c’;

int s_total;

s_total = k + ch;

printf(“s_total = %d\n”, s_total );

}
```

At the point when the above code is compiled and executed, it creates the accompanying result:

```
Value of s_total : 104
```

The result is in view of the fact that the ASCII equivalent of ‘c’ is 99. When 5 is added to 99, the final value becomes 104.

Normal Arithmetic Conversion

The normal mathematical changes are certainly performed to cast their values in a typical data type. Compiler first performs integer promotion. If operands still have distinctive

sorts, then they are changed over to the data type that seems most noteworthy in the accompanying order:

The common number-crunching transformations are not performed for the task operators, nor for the logical administrators && and ||. Given us a chance to take emulating sample to comprehend the idea:

```
#include <stdio.h>

int main () {

int  k = 5;

char ch = ‘c’;

float final;

final = ch + k;

printf(“final = %f\n”, final );

}
```

At the point when the above code is compiled and executed, it delivers the accompanying result:

final = 104.000000

Here, it is easy to comprehend that first c gets changed over to integer. However, since last value is double so ordinary math change applies and compiler changes over k and ch into float data type. Finally, this value is assigned to the float variable final.

Input & Output

When we use the term input, we intends to speak of any information that is being sent into the system. This can be given as record or from command from the console. C programming language gives a set of functions, as part of its standard library, to get input

and process the same.

On the other hand, when we talk about output, we intend to show some information on screen, printer or in any file. C programming language gives a set of implicit functions to output the information on the machine screen and you can save this information in a file or pool it to the printer.

The Standard Files

C programming language treats all the input devices as files. So devices, for example, the keyboard are attended to in the same manner as files. In the same manner, the printer and console are also treated in the same manner. Here is a list of devices and the corresponding file structure.

- Standard File - file Pointer
- Gadget - stdin
- Standard input - stdin
- Console - stdin
- Standard output - stdout
- Screen - stdout
- Standard error - stderr

The Functions - getchar() & putchar()

The function getchar() peruses the following character from the screen and returns it as an integer. This function peruses just one single character at once. You can utilize this strategy as a part of the loop on the off chance that you need to peruse more than one character from the screen.

The function putchar() puts the given character on the screen. This function puts just

single character at once. You can utilize this strategy as a part of the loop on the off chance that you need to show more than one character on the screen.

The following code illustrates the use of these functions:

```
#include <stdio.h>

int main ( )

{

int x;

printf( “Enter a number:”);

x = getchar( );

printf( “\nNumber entered: “);

putchar( c );

return 0;

}
```

At the point when the above code is compiled and executed, you will realize that the system waits for you to enter a character. When you enter a character and press enter, then the program reads the characters and then, writes the character on the screen.

The Functions - gets() & puts()

The function char *gets(char *str) peruses a line from stdin into the array indicated by str until either an ending newline or EOF. On the other hand, the function int puts(const char *s) writes the string str and a trailing newline to stdout. The functioning of these inbuilt functions are illustrated in the following code:

```
#include <stdio.h>
```

```
int main( )

{

char my_str[50];

printf( “Enter text:”);

my_str = gets( my_str );

printf( “\nText received: “);

puts( my_str );

return 0;

}
```

At the point when the above code is compiled and executed, the system waits for you to enter some text and press enter. When you enter some text and press enter, the program reads it and writes the same onto the screen.

The Functions - scanf() and printf()

The function int scanf(const char *formatstr, …) peruses information from the standard stream of data, stdin. On the other hand, the function int printf(const char *formatstr, …) writes text to the standard output stream, stdout. The formatstr specified in the given function declaration gives %d, %s, %f, %c, and so on. These strings are used to print or read strings, characters, integers or float numbers. The following code illustrate the use of these functions:

```
#include <stdio.h>

int main ( )

{

char my_str[50];
```

```
int x;

printf( “Enter value:”);

scanf(“%d”, &x);

printf( “\nEntered number = %d “, x);

return 0;

}
```

At the point when the above code is compiled and executed, it holds up for you to enter something. When you enter an integer and press enter, the program returns and peruses the data. Consequently, the same text is printed onto the screen.

Here, it ought to be noted that scanf () expects the same type of data that the format string mentions. In the event that you give “string” instead of an integer, it shall be considered a wrong entry. Second, while perusing a string scanf (), the system quits as soon as it experiences a space. Therefore, when you write a b c, then a, b and c are three different strings fro scanf ().

File Input and Output

Last section clarified about standard input and output functions provided by C programming dialect. This part we will perceive how C software developers can create, open, close and edit files.

A file is a grouping of bytes. However, please remember that every input or output device is a file for C and files are no difference. C programming language gives access to low level (OS level) calls to handle files. This part will take you through essential functions and programming constructs required for file handling.

Opening Files

You can utilize the function fopen ( ) to make another file or to open a current file of the specified name and in the specified mode. This call will instate an object of the type FILE, which contains all the data required for controlling the file stream. The function call can be made in the following manner:

File *fopen(const char * samplefile, const char * filemode);

Here, samplefile is string containing the name of the file, which you will use to name your file. The second parameter in the function call is the access mode, which can take one of the following values:

- r

  Opens a current file for reading purposes only.

- w

  Opens a file for writing. However, in the event that the file doesn’t exist, then a file with the specified name is made. Here your system will begin composing content from the earliest starting point of the record.

- a

  Opens a file for writing in adding mode. However, in the event that the file doesn't exist, another record is made. Here your system will begin appending the file from the place where the already written content ends.

- r+

  Opens a file for both writing and reading both.

- w+

  Opens a file for writing and reading both. It first truncates the file to zero length on the off chance that it exists. However, if there is no file, a new file is created.

- a+

  Opens a file for writing and reading both. It makes the document on the off chance that it doesn't exist. The reading will begin from the earliest starting point yet writing must be attached.

On the off chance that you are going to handle binary files, you will use beneath specified access modes rather than the aforementioned:

"wb", "rb", "rb+", "ab", "wb+", "r+b","ab+", "w+b" and "a+b"

Shutting a File

To close a record, utilize the function fclose ( ). The syntax of this function call is as follows:

int fclose(FILE *fptr);

How To Write Into Files

In order to write to files, the following functions are used,

int fputs(const char *str, FILE *fptr);

int fputc(int ch, FILE *fptr);

The function fclose( ) return 0 on success and EOF if some error occurs.

This function, flushes any information as of now pending in the file array, shuts the file, and discharges any memory utilized for the document. The EOF is declared and defined in the header file, stdio.h.

There are different functions given by C standard library to peruse and compose a file, character by character or as a line. The accompanying segments discusses some of these functions:

The fputc() function composes the character value of the argument ch to the out stream referenced by fptr. It furnishes a proportional payback character composed on success and EOF if there is a lapse.

The fputs() function writes the string s to the out stream referenced by fptr. It gives back a non-negative value on success and EOF is returned if there should arise an occurrence of any lapse. You can utilize int fprintf(file *fptr, const char *formatptr, …) work too to write a string into a file. Attempt the accompanying code:

```
#include <stdio.h>

int main () {

File *fptr;

fptr = fopen(“sample.txt”, “w+”);

fprintf(fptr, “fprintf executing..\n”);

fputs(“fputs executing..\n”, fptr);

fclose(fptr);

}
```

At the point when the above code is compiled and executed, it makes another file sample.txt in the same directory as the file.

How To Read Files

In order to read files, the following two functions are commonly used:

int fgetc(FILE * fptr);

char *fgets(char *buffer, int count, FILE *fptr);

The function fgetc() peruses a character from the file referenced by fptr. The return value is the character read, or if there should be an occurrence of any lapse it returns EOF. The function fgets() peruses up to (count – 1) characters from the data stream referenced by fptr. It duplicates the read string into the array buffer, affixing an invalid character to end the string.

In the event that this function experiences a newline character “\n” or the end of the file, EOF, before they have perused the greatest number of characters, then it returns just the characters read up to that point including new line character. You can likewise utilize int fscanf(file *fptr, const char *formatstr, …) to peruse strings from a file. However, it quits perusing after the first space character is encountered.

#include <stdio.h>

int main () {

File *fptr;

char buffer[50];

fp = fopen(“sample.txt”, “r”);

fscanf(fp, “%s”, buffer);

printf(“1 : %s\n”, buffer);

```
fgets(buffer, 255, (File*)fptr); printf(“2: %s\n”, buffer );

fgets(buffer, 255, (File*)fptr); printf(“3: %s\n”, buffer );

fclose(fptr);

}
```

Binary I/O Functions

There are two functions, inbuilt in the C standard library, which can be utilized for binary input/output:

```
size_t fread(void *s_ptr, size_t element_size, size_t elements_num, FILE * file_ptr);

size_t fwrite(const void *s_ptr, size_t elements_size, size_t elements_num, FILE * file_ptr);
```

Preprocessors

The C Preprocessor is not a part of the compiler. However, preprocessing is a part of the assembling process. In oversimplified terms, a C Preprocessor is simply a content substitution tool, which educates the compiler to do obliged preprocessing. Any preprocessor summons starts with a hash special character (#).

It must be the first nonblank character, and for decipherability, a preprocessor mandate ought to start in first segment of the code. Some of the preprocessor mandates that are available in the C programming language are as follows:

- #define

  Any macro defined using this directive is simply substituted in the code by the preprocessor.

- #include

  The file included using this directive is added to the project for compilation and linking. This directive must ideally appear in the first part of the code.

- #undef

  This directive is used for un-defining any of the defined directives.

- #ifdef

  This directive is a conditional directive and returns true if this macro is defined.

- #ifndef

  This directive is a conditional directive and returns false if this macro is defined.

- #if

  This directive is a conditional directive and tests for macro definition.

- #else

This directive is a conditional directive and follows if directive.

- #elif

This directive is a conditional directive and is a combination of #if and #else.

- #endif

This directive is a conditional directive and is used for closing #if.

- #error

This directive is used for error reporting.

- #pragma

This directive issues exceptional summons to the compiler, utilizing an institutionalized strategy.

Preprocessors Examples

Here are a few examples of the preprocessor directives for your reference and understanding.

- #define MAXVAL 5

This order advises the C preprocessor to substitute any occurrences of the macro MAXVAL with the integer value 5. This type of macro definition is used commonly for defining constants.

- #include <stdio.h>

These mandates advise the C preprocessor to get stdio.h from System Libraries and add the content to the current source file. This is a standard way of including any header files from the standard C library.

- #include “myheader.h”

The line advises C preprocessor to get myheader.h from the neighborhood registry and add the content to the current source file. This method of file inclusion is commonly used for including user created header files, which are present at the same location as the source files concerned.

Predefined Macros

Some macros are predefined in the C library. A few of these macros are MESSAGE and DEBUG. Their usage and syntax is given below:

Example 1:

This example tells the preprocessor that it must define MESSAGE as “Here I am!” if MESSAGE has not already been defined.

```
#ifndef MESSAGE

#define MESSAGE “Here I am!”

#endif
```

Example 2:

These statements tell the preprocessor that if the DEBUG macro is defined, then the following statements that occur after the #ifdef must be executed.

```
#ifdef DEBUG

//Debugging statements

#endif
```

In addition to these, the C programming environment also provides some other macros like the ones mentioned below:

- __time__

This macro defines the current time in the following format -

“HH:MM:SS”.

- __date__

This macro defines the current date in the following format - “MMM DD YYYY”.

- __line__

This macro defines the current line number.

- __file__

This macro defines the current filename.

- __stdc__

If the compiler is following standard ANSI, the value of this macro is set to 1.

Preprocessor Operators

- Macro Continuation (\)

Macro definitions must ideally end on the same line on which they start. However, in case of parameterized macros, which will be discussed in the next section, you may need more than one line to define the macro. If your macro is continuing on the next line, the previous line must end with the macro continuation operator (\).

- Stringize (#)

Whenever this operator is used within a macro definition, it is used for converting any parameter of the macro into a constant of string type.

- Token Pasting (##)

This operator performs the combination function. Therefore, if this operator is used with two operators in a macro definition, these arguments shall be converted

into a single argument.

- The defined() Operator

This operator is provided to help the developer ensure and check whether an identifier has a matching macro definition for it. This function returns true if a macro definition for concerned identifier exists.

Parameterized Macros

One of the influential capacities of the C preprocessors is the capacity to mimic functions utilizing parameterized macros. Here is an example of such a construct:

int square_num(int i) { return i * i;}

Macros with arguments must be defined utilizing the #define before they can be utilized. The arguments list is encased in brackets and must instantly follow the macro name. Spaces are not permitted between and macro name and open bracket.

Java For Beginners

A Simple Start To Java Programming (Written By A Software Engineer)

Scott Sanderson

Table of Contents

Introduction

Java, the programming language, was introduced by Sun Microsystems. This work was initiated by James Gosling and the final version of Java was released in the year 1995. However, initially Java was released as a component of the core Sun Microsystem platform for Java called J2SE or Java 1.0. The latest release of Java or J2SE is Java Standard Version 6.

The rising popularity of Java, as a programming platform and language has led to the development of several tools and configurations, which are made keeping Java in mind. For instance, the J2ME and J2EE are two such configurations. The latest versions of Java are called Java SE and Java EE or Java ME instead of J2SE, J2EE and J2ME. The biggest advantage of using the Java platform is the fact that it allows you to run your code at any machine. So, you just need to write your code once and expect it to run everywhere.

As far as the features of Java are concerned, they are as follows:

- Object Oriented

  In Java, everything is an object. Java can be effectively stretched out and extended to unimaginable dimensions since it is focused around the Object model.

- Independent of the platform

  Dissimilar to numerous other programming dialects including C and C++, when Java is aggregated, it is not converted into a form, which is particular to any

machine. Instead, it is converted into a machine-independent byte code. This byte code is conveyed over the web and deciphered by Virtual Machines or JVM on whichever stage it is generally run.

- Simple

  Java is intended to be not difficult to learn. In the event that you comprehend the essential idea of OOP, Java would not be difficult to ace.

- Secure

  With Java's security framework, it empowers to create frameworks, which are free of viruses and tampering. Public-key encryption is used as the core authentication strategy.

- Independent of Machine Architecture

  Java compiler produces an object file format, which is independent of the architecture of the machine. The assembled code can be executed on numerous processors, with the single requirement that they must all have Java runtime framework.

- Portability

  The fact that Java code is machine and platform independent makes it extremely compact. Compiler in Java is composed in ANSI C with a clean conveyability limit, which is a POSIX subset.

- Robustness

  Java tries to kill circumstances, which can lead to potential system failures, by stressing chiefly on runtime checking and compile time checking.

- Support for Multithreaded Applications

With Java's multithreaded feature, it is conceivable to compose programs that can do numerous assignments at the same time. This configuration gimmick permits designers to build easily running intelligent applications.

- Interpreted Code

  Java byte code is interpreted on the fly to local machine. The advancement methodology is more quick and expository since the interfacing is an incremental and lightweight process.

- High Performance

  With the utilization of Just-In-Time compilers, Java enhances the performance of the system.

- Distributed

  Java is intended for the conveyed environment of the web.

- Dynamic

  Java is thought to be more dynamic than C or C++ since it is intended to adjust to an advancing environment. Java projects can convey broad measure of run-time data that can be utilized to check for accesses and respond to the same on run-time.

History of Java

James Gosling started working on the Java programming language in June 1991 for utilization in one of his numerous set-top box ventures. The programming language, at first, was called Oak. This name was kept after an oak tree that remained outside Gosling's office. This name was changed to the name Green and later renamed as Java, from a list of words, randomly picked from the dictionary.

Sun discharged the first open usage as Java 1.0 in 1995. It guaranteed Write Once, Run Anywhere (WORA), giving no-expense run-times on prominent stages. On 13 November

2006, Sun discharged much of Java as free and open source under the terms of the GNU General Public License (GPL). On 8 May 2007, Sun completed the procedure, making the greater part of Java's center code free and open-source, beside a little parcel of code to which Sun did not hold the copyright.

Pre-requisites

In order to run and experiment with the examples given in this book, you shall require a Pentium 200-Mhz machine with at least 64 MB of RAM. You additionally will require the accompanying programming platforms:

- Microsoft Notepad or Any Word Processor
- Java JDK 5
- Linux 7.1 or Windows XP or higher Operating Systems

Java - Basic Syntax

A basic Java program can be broken down into several constructs and elements. Typically, it can be characterized as a collection of objects, which communicate with each other by calling each other's routines. The basic definitions of objects and classes are given below:

- Class

  A class can be described as a blueprint that portrays the practices/expresses all the behaviors and states of its objects.

- Object

  Objects are characterized by two components namely, methods and attributes or variables. For instance, if you consider the example of a puppy, then it has the following attributes or states: name, color and breed. In addition, it also has the following behaviours, which include woofing, wagging and consuming. Any object is nothing but an instance of a class.

- Instance Variables

  Each object has its set of variables. An object's state is made by the qualities alloted to these variables during program execution.

- Methods

  A method is the definition of a method. Moreover, a class can contain numerous methods. It is in these methods that behaviours like where the rationales are composed, information is controlled and all the activities are executed.

First Java Program:

In order to start with basic basic Java programming, let us look at the standard Hello World program.

```
public class MyFirstJavaProgram {

        public static void main(String []args) {

        System.out.println(“Say Hello World To The World!”);

        }

}
```

As you can see, the program uses a single line of code in the main() function, which prints the statement ‘Hello World!’. However, before that can be done, let us look at the steps that you must follow in your quest to execute the file.

- Open any text editor and paste this code in that file.
- Save the file with a .java extension. For example, you can save the file as Sample.java.
- The next step is to to open the command prompt of the system and relocate its reference to the directory in which the file is saved. For instance, if you have saved the file in C:\, then you must take the prompt to the same directory.
- In order to compile the code, you must type the following:

  javac Sample.java

- If there are no errors, you will automatically be taken to the next line. You can now execute the code using the following command:

  java Sample.java

- You should be able to see the following output on the screen.

  Say Hello World To The World!

Basic Syntax

About Java programs, it is paramount to remember the accompanying points.

- Class Names –

  For all class names, the first letter ought to be in Upper Case.

  On the off chance that few words are utilized to structure a name of the class, every internal word's first letter ought to be in Upper Case. For example, a standard class name is as follows:

  class Sampleclass

- Case Sensitivity - Java is case sensitive, which implies that the identifier Hi and hi would have distinctive importance in Java.
- Method Names - All system names ought to begin with a Lower Case letter. In the event that few words are utilized to structure the name of the method, then every internal word's first letter ought to be in Upper Case. An example of this convention is follows:

  public void mysamplemethod ()

- Filename –

  The name of the system record ought to precisely match the class name. At the point when you are saving the file, you ought to save it utilizing the class name. Remember Java is case touchy and affix ".java" to the end of the name. If the document name and the class name don't match your system won't assemble. Consider the example of a class name Sample. In this case, you must name the file as sample.java.

- public static void main(string args[])

  Java system handling begins from the main() function, which is a required piece of each Java program.

Java Identifiers

All Java components require names. Names utilized for classes, variables and strategies are called identifiers. In Java, there are a few focuses to recall about identifiers. They are as per the following standard:

- All identifiers ought to start with a letter (beginning to end or a to z), underscore (_) or special character ($).
- After the first character, identifiers can have any mix of characters.
- You cannot use a keyword as an identifier.
- Most significantly, identifiers are case sensitive. So, Sample is not same as sample.
- Examples of identifiers include $salary, age, __1_value and _value.
- Examples of illicit identifiers include –compensation and 123abc.

Java Modifiers

Like is the case with any programming language, it is conceivable to alter classes and systems by utilizing modifiers. There are two classifications of modifiers:

- Access Modifiers: public, default, protected and private
- Non-access Modifiers: strictfp, final and abstract

We will be researching more insights about modifiers in the following chapters.

Java Variables

Several types of variables are supported by Java. These types of variables include:

- Instance Variables (Non-static variables)
- Class Variables (Static Variables)
- Local Variables

Java Arrays

Arrays are contiguous memory locations that store different variables of the same sort. On

the other hand, an array itself is an article on the memory. We will research how to proclaim, develop and instate these in the chapters to follow.

Java Enums

Enums were introduced as part of the Java package in java 5.0. Enums limit a variable to have one of just a couple of predefined qualities. The qualities in this identified list are called enums. With the utilization of enums it is conceivable to diminish the quantity of bugs in your code. Case in point, in the event that we consider an application for a cafe, it would be conceivable to limit the mug size to extra large, large, medium and small. This would verify that it would not permit anybody to request any size other than the sizes mentioned in the menu or application listing.

Please note that enums can be pronounced as their own or inside a class. However, routines, variables, constructors can be created inside the body of enums as well.

Java Keywords:

Keywords or reserved words in Java are shown in the table below. As a rule, these words cannot be used as names for variables or constants.

- assert
- abstract
- break
- boolean
- case
- byte
- char
- catch
- const
- class

- default
- continue
- double
- do
- enum
- else
- final
- extends
- float
- finally
- goto
- for
- implements
- if
- instanceof
- import
- int
- long
- interface
- new
- native
- private
- package
- protected
- return
- public

- static
- short
- super
- strictfp
- synchronized
- switch
- throw
- this
- transient
- throws
- while
- try
- volatile
- void

Comments in Java

Just as in the case of C++ and C, Java supports two types of comments namely, single line comments and multi-line comments. The syntax for these types of comments are as follows:

Single line comment:

//<comment>

Multiple line comment:

/*<comment>*/

All characters that exist in the comments region are simply ignored by the Java compiler.

Using Blank Lines:

Any line that is only composed of whitespace characters or comments is considered a blank line. These lines are just ignored by the compiler and are not included in the executable.

Inheritance:

Java supports inheritance. In other words, it is possible to derive one class from another class in this programming language. For instance, if you need to create a new class, which is an extension of an existing class, then you can simply derive this new class from an existing class. This allows the new class to access the elements already implemented in the existing class. In this case, the new class is called the derived class and the existing class is referred to as the super class.

Interfaces:

As mentioned previously, Java is all about interaction between objects. The manner in which different objects communicate with each other is defined in what is called an ‘interface.’ Moreover, interfaces are also an important aspect of the inheritance feature of java. As part of an interface, the methods that can be used by a derived or sub-class are declared. However, all the methods declared as usable for the subclass must be implemented in the subclass.

Objects and Classes

Java is an Object-Oriented programming language. As an issue that has the Object Oriented peculiarity, Java underpins the accompanying essential ideas:

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation
- Objects
- Message Parsing
- Classes
- Method
- Instance

In this part, we will investigate the concepts of Classes and Objects.

- Class - A class can be described as an blueprint that declares and defines the attributes and methods that its objects will implement and use.
- Object - Objects are simple real world entities that possess a state and its characteritic behaviour.

For example, if you consider a real world entity, a labrador dog, then this dog is an object. However, it belong to the class of dogs. Therefore, the associated class is Dog.

Objects in Java

Let us now look profoundly into what are objects. In the event that we consider this present reality, we can discover numerous entities around us, Cars, Humans, Dogs and several other. N fact, any real world entity can be modelled as an object. The one common

thing between all these entities is the fact that they contain states and behaviours. On the off chance that we consider a dog, then its state is - breed, name and color. However, its behaviour includes eating habits and other characteristics like running and barking.

Classes in Java

A class is a blue print from which individual objects are made. A specimen of a class is given underneath:

```
open class Dogs {

String breed;

String shade;

int age;

void eating (){  }

void barking (){  }

}
```

A class can contain any of the accompanying variable sorts.

- Local variables

  Variables that are declared and used inside routines, constructors or pieces of code are called local variables. The variable will be proclaimed and instated inside the method or scope and the variable will be destroyed when the execution of a method terminates.

- Instance variables

  Instance variables are variables inside a class yet outside any system. These variables are instantiated when the class is stacked. These variables can be gotten to from inside any technique, constructor or squares of that specific class.

- Class variables

  Class variables will be variables, which are declared within a class, outside any system, with the static word before them.

A class can have any number of routines to get to the estimation of different sorts of methods. In the above illustration, eating() and barking() are the used methods. Underneath specified are a percentage of the vital subjects that need to be examined when researching classes of the Java Language.

Constructors

At the point when talking about classes, a standout amongst the most vital sub theme would be constructors. Each class has a constructor. In the event that we don’t unequivocally compose a constructor for a class, the Java compiler manufactures a default constructor for that class. Each time an object is made, no less than one constructor will be summoned.

The fundamental principle of constructors is that they ought to have the same name as the class. A class can have more than one constructor and depending on the parameters given and return type expected, the matching constructor is called. A sample implementation for this type of a method is given below:

```
public class Puppies{

public Puppies(){

}

public Puppies(string puppyname){

}
```

The above class has two constructors. One of the constructors requires no parameters. However, the other constructor requires a string equivalent to the name of the puppy. Java

additionally upholds Singleton Classes where you would have the capacity to make one and only object of a class.

Making Objects

As specified previously, a class gives the outlines to object creation. So, fundamentally an object is made from a class. In Java, the new essential word is utilized to make new objects.

There are three steps involved in the creation of any object. These steps are illustrated below:

- Declaration: A variable assertion with a variable name and object type.
- Instantiation: The “new” word is utilized to make the object of an already declared class.
- Initialization: The “new” word is trailed by a call to a constructor. This call instantiates the class and creates an object of the same, as a result.

Sample implementation is given below for better understanding of the concept.

```
public class Puppies{

public Puppies(string name){

        System.out.println(“Passed Name of the puppy is:” + name );

}

public static void main(string []args){

        Puppies samplepuppy = new Puppies( “jimmy” );

}
```

On the off chance that we compile and run the above project, then it would deliver the accompanying result:

Passed Name of the puppy is: jimmy

Getting to Instance Variables and Methods:

Variables and methods are gotten to by means of made objects of classes. To get to a variable, the qualified way ought to be the following:

The following statement creates an object.

Newobject = new Constructormethod();

The following statements can be used to access the variable and method associated with the object.

Newobject.variablesname;

Newobject.methodsname();

A sample implementation of this concept is given below:

```
public class Dog{

int dogAge;

public dogAge(String dogname){

        System.out.println(“Dog Name Passed is :” + dogname );

}

public void initAge( int dogage ){

        dogAge = dogage;

}

public int getDogAge( ){

        System.out.println(“Dog’s present age is:” + dogAge );

        return dogAge;
```

```
}
public static void main(String []args){
        Dog myDog = new Dog( “jimmy” );
        myDog.initAge( 5 );
        myDog.getDogAge( );
        System.out.println(“Variable dogAge Value is:” + myDog.dogAge );
}
}
```

Upon compilation and execution of the following code, you shall be able to see the following result.

Variable dogAge Value is: 5

Declaration Guidelines for Source Files

As the last piece of this area how about we now investigate the source file declaration standards. These tenets are key when declaring classes, importing declarations and packages in a source file.

- There can be stand out public class for every source record.
- A source document can have numerous non public classes.
- The public class name ought to be the name of the source document. The name of the source file must be affixed by the string .java. For instance, the class name is public class Employeerecord{}, then the source document ought to be saved as Employeerecord.java.
- If the class is declared inside a package, then the package articulation ought to be the first proclamation in the source record.

- If import articulations are available, then they must be composed between the package proclamation and the class revelation. On the off chance that there are no package proclamations, then the import articulation ought to be the first line in the source file.
- Import and package articulations will intimate to all the classes show in the source record. It is impractical to announce diverse import and/or package explanations to distinctive classes in the source file.

Classes have a few access levels. Moreover, there are diverse sorts of classes, which include final classes, in addition to several others. Separated from the aforementioned sorts of classes, Java likewise has some uncommon classes called Inner classes and Anonymous classes.

Java Packages

Basically, it is a method for classifying the classes and interfaces. At the point when creating applications in Java, many classes and interfaces will be composed. In such a scenario, ordering these classes is an unquestionable requirement and makes life much less demanding. In Java, if a completely qualified name, which incorporates the class and package name, is given, then the compiler can without much of a stretch find the source code or classes. Import declarations is a method for giving the correct area for the compiler to find that specific class.

Case in point, in order to load all the classes accessible in java_installation/java/io, you must use the following statement:

import java.io.*;

A sample implementation for this concept is given below:

The following code uses two classes Employeerecord and Employeerecordtest. The first

step is to open the text editor you intend to use at your system and copy and paste the code shown below into the text editor application. Keep in mind that the public class in the code is Employeerecord. Therefore, the name of the file should be Employeerecord.java. This class uses the variables, methods and constructor as shown below:

```
import java.io.*;

public class Employeerecord {

int empage;

String empname;

double empcompensation;

public Employee(string empname){

this.empname = empname;

}

public void employeeage(int employeeage){

empage =  employeeage;

}

public void empcompensation(double empcompensation){

empcompensation = empcompensation;

}

public void printemp(){

System.out.println(“empname:”+ empname );

System.out.println(“empage:” + empage );

System.out.println(“empcompensation:” + empcompensation);
```

}

As specified awhile ago in this exercise, handling begins from the main function. Accordingly, with this goal, we should create a main function for this Employeerecord class. Given beneath is the Employeerecordtest class, which makes two instances of the class Employeerecord and conjures the techniques for each one item to allot values for every variable. You can save this file as Employeerecordtest.java.

```
import java.io.*;

public class Employeerecordtest{

public static void main(String args[]){

Employeerecord employee1 = new Employeerecord(“Jack Wright”);

Employeerecord employee2 = new Employeerecord(“Mary John”);

employee1.employeeage(32);

employee1.empcompensation(5000);

employee2.employeeage(25);

employee2.empcompensation(2000);

employee1.printemp();

employee2.printemp();

}

}
```

Upon compilation and execution, you must get the following output:

```
empname: Jack Wright

empage: 32
```

empcompensation: 5000

empname: Mary John

empage: 25

empcompensation: 2000

Basic Data Types

Variables are only saved memory areas to store values. This implies that when you make a variable, you save some space in memory. In light of the data type of a variable, the working framework distributes memory and chooses what can be put in the held memory. Consequently, by appointing diverse data types to variables, you can store whole numbers, decimals, or characters in these variables.

There are two data types accessible in Java:

- Reference/Object Data Types
- Primitive Data Types

Primitive Data Types

There are eight primitive information types, which are supported by Java. Primitive data types are predefined by the dialect and named by a catchphrase. This section discusses these data types in detail.

byte:

- byte information sort is a 8-bit marked two's supplement whole number.
- Maximum worth is 2^7 -1, which is equal to 127. This value is also included in the range of these values.
- Minimum worth is -2^7, which is equal to -128.
- Default value stored in a variable of this type is 0.
- byte information sort is utilized to spare space in vast exhibits, principally set up of numbers, since a byte is four times littler than an int.
- Example:

  byte x = 200, byte y = -20

short:

- short information sort is a 16-bit marked two's supplement number.
- Maximum value is 2^15 -1, which is equal to 32,767. This number is also included in the range.
- Minimum value is -2^15, which is equal to -32,768.
- short information sort can likewise be utilized to spare memory as byte information sort. A short is 2 times littler than an int
- The default value for this data type is 0.
- Example:

  short x = 425164, short y = -76686

int:

- int information sort is a 32-bit marked two's supplement number.
- Maximum value for this data type is 2^31 -1, which is equal to 2,147,483,647. This number is also included in the range for this data type.
- Minimum value for this data type is -2^31, which is equal to - 2,147,483,648.
- int is for the most part utilized as the default information sort for its indispensable qualities unless there is a worry about memory.
- The default value for this data type is 0.
- Example:

  int x = 826378, int y = -64782

long:

- long information sort is a 64-bit marked two's supplement whole number.
- Maximum value for this data type is 2^63 -1, which is equal to 9,223,372,036,854,775,807.

- Minimum value for this data type is -2^63, which is equal to -9,223,372,036,854,775,808.
- This sort is utilized when a more extensive memory range than int is required.
- The default value for those data type is 0l.
- Example:

  long x = 174636l, int y = -536452l

float:

- float is a data type, which is know for its solitary exactness, 32-bit IEEE 754 gliding point.
- float is for the most part used to spare memory in vast exhibits of coasting point numbers.
- The default value for this data type is 0.0f.
- float information sort is never utilized for exact values, for example, money.
- Example:

  float x = 254.3f

double:

- double information sort is a float with twofold exactness 64-bit IEEE 754 drifting point.
- This information sort is for the most part utilized as the default information sort for decimal qualities.
- double information sort ought to never be utilized for exact values, for example, money.
- The default value for this data type is 0.0d.
- Example:

double x = 321.4

boolean:

- boolean information sort speaks to one bit of data.
- Any boolean variable can assume one of the two values: true or false.
- This information sort is utilized for basic banners that track genuine/false conditions.
- The default value for this data type is false.
- Example:

boolean check = true;

char:

- char information sort is a solitary 16-bit Unicode character.
- Maximum value for a variable of this type is “\uffff” (or 65,535 comprehensive).
- Minimum value for a variable of this type is “\u0000” (or 0).
- char information sort is utilized to store any character.
- example:

char text =‘a’

Reference Data Types

- Reference variables are made utilizing characterized constructors of the classes. They are utilized to get to objects. These variables are proclaimed to be of a particular data type that can’t be changed. A few examples of such data types are Employee and Dog.
- Class objects, and different kind of variables go under reference data type.
- Default estimation of any reference variable is invalid.
- A reference variable can be utilized to allude to any object of the announced sort.

- Example: myanimal = new Animals(“rabbit”);

Java Literals

A literal in Java is a source code representation of a settled worth. They are spoken to specifically in the code without any calculation. Literals can be appointed to any primitive sort variable. Case in point:

byte x = 86;

char x = “a”

int, byte, short and long can be communicated in hexadecimal(base 16), decimal(base 10) or octal(base 8) number frameworks too. Prefix 0 is utilized to show octal while prefix 0x demonstrates hexadecimal when utilizing these number frameworks for literals. For example,

int numd = 134;

int numo = 0243;

int numx = 0x95;

String literals in Java are determined like they are in most different programming languages by encasing a grouping of characters between a couple of twofold quotes. Illustrations of string literals are:

“Hi Everyone” “two\nlines” “\“these characters are inside quotes\””

String sorts of literals can contain any Unicode characters. For instance:

String news = “\u0001”

You can also use escape sequences with Java. Here is a list of escape sequences that you can use.

Double quote - \”

Carriage return (0x0d) - \r

Newline (0x0a) - \n

Single quote - '

Backspace (0x08) - \b

Formfeed (0x0c) - \f

Tab - \t

Space (0x20) - \s

Octal character (ddd) - \ddd

Backslash - \

Hexadecimal UNICODE character (xxxx) - \uxxxx

Variable Types

A variable gives us named capacity that our code can control. Every variable in Java has a particular sort, which decides the size and format of the variable's memory; the scope of values that can be put away inside that memory; and the set of operations that can be connected to the variable. You must make an explicit declaration of all variables before they can be utilized. Variables can be declared in the following manner:

Data type <variable name>;

Here data type is one of Java's datatypes. On the other hand, a variable is the name or the identifier associated with the variable. To pronounce more than one variable of the pointed out type, you can utilize a comma-divided rundown. Here are a few examples of declarations:

The following declaration declares three integer variables.

int x, y, z;

In a similar manner, variables of other data types may also be declared.

Java supports three types of variables. These types are as follows:

- Class/static variables
- Instance variables
- Local variables

Local Variables

- Local variables are announced in systems, constructors, or scopes.
- Local variables are made when the constructor or method is entered and the variable will be decimated once it retreats the system, constructor or scope.
- Access modifiers can't be utilized for neighborhood variables.

- Local variables are noticeable just inside the announced method, constructor or scope.
- Local variables are executed at stack level.
- There is no default value for these variables. So, local variables ought to be declared and a beginning value ought to be relegated before the first utilization.

Sample Implementation:

Here, age is a neighborhood variable. This is characterized inside pupage() strategy and its degree is constrained to this system just.

```
public class myTest{

open void newfunc(){

int myvar = 1;

myvar = myvar + 10;

System.out.println(“The value of myvar is: ” + myvar);

}

public static void main(string args[]){

mytest = new myTest ();

mytest.myfunc();

}
```

The output of the execution of this code is:

The value of myvar is: 11

Instance Variables

- The declaration of an instance variable is made inside the class. However, it is

made outside the system, constructor or any scope.

- Instance variables are made when an object is made with the utilization of the keyword “new” and obliterated when the item is destroyed.
- When a space is dispensed for an item in the memory, an opening for each one variable value is made.
- Instance variables can be pronounced in class level before or after utilization.
- Instance variables hold values that must be referenced by more than one method, constructor or piece, or key parts of an object’s express that must be available all through the class.
- Access modifiers can be given for sample variables.
- Instance variables have default values. For numbers, the default quality is 0. However, for Booleans, it is false and for object references, it is invalid. Qualities can be relegated amid the statement or inside the constructor.
- The case variables are unmistakable for all methods, constructors and scope in the class. Regularly, it is prescribed to make these variables private (access level). However perceivability for subclasses can be given with the utilization of access modifiers for these variables.
- Instance variables can be gotten to by calling the variable name inside the class. The following statement can be used for this purpose: Objectreference.variablename.

Sample Implementation:

```
import java.io.*;

public class Employeerecord{

public String empname;

private double empcompensation;
```

```
public Employee (String name){

empname = name;

}

public void initsalary(double empsalary){

empcompensation = empsalary;

}

public void printemployee(){

System.out.println(“Employee name  : ” + empname );

System.out.println(“Employee salary :” + empcompensation);

}

public static void main(string args[]){

Employeerecord employee1 = new Employeerecord(“Mary”);

employee1.initsalary(7500);

employee1.printemployee();

}
```

The compilation and execution would deliver the accompanying result:

```
Employee name  : Mary

Employee compensation :7500.0
```

Class/Static Variables

- Class variables otherwise called static variables are declared with the static keyword in a class, yet outside a constructor, method or scope.
- There would just be one duplicate of each class variable for every class, paying

little mind to what number of objects are made from it.

- Static variables are seldom utilized other than being pronounced as constants. Constants are variables that are announced as private/public, static and final. Consistent variables never show signs of change from their introductory quality.
- Static variables are put away in static memory. It is uncommon to utilize static variables other than announced final and utilized as either private or public constants.
- Static variables are made when the system begins and annihilated when the execution stops.
- Visibility is like instance variables. In any case, most static variables are announced public since they must be accessible for clients of the class.
- Default values for these variables are also same as instance variables. For numbers, the default value id typically 0. However, the same value for Booleans is false and for object reference is invalid. Values can be doled out amid the assertion or inside the constructor. Furthermore, values can be appointed in unique static initializer brackets.
- Static variables can be gotten to by calling with the class name . Classname.variablename.
- When announcing class variables as public static final, variables names (constants) must all be in upper case. Moreover, the static variables are not public and the naming convention is the same as local and instance variables.

Sample Implementation:

```
import java.io.*;

public class Employeerecord{

private static double empcompensation;
```

```
public static final String empdept = “HR “;

public static void main(string args[]){

empcomp = 7500;

System.out.println(empdept+” Compensation: “+empcompensation);

}
```

The compilation and execution of this code shall create the accompanying result:

HR Compensation: 7500

Modifier Types

Modifiers are catchphrases that you add to definitions to change their implications. The Java programming language has a wide and mixed bag of modifiers, including the accompanying:

- Non-Access Modifiers
- Java Access Modifiers

In order to utilize a modifier, you incorporate its catchphrase in the meaning of a class, variable or method. The modifier goes before whatever is left of the announcement.

Access Control Modifiers:

Java gives various access modifiers to set access levels for classes, variables, routines and constructors. The four right to gain access are:

- Private: visible to the class.
- Default: visible to the bundle. No modifiers are required.
- Secured: visible to all subclasses and package.
- Public: visible to the world.

Non Access Modifiers:

Java gives various non-access modifiers to attain numerous other usefulness.

- Static:

  The static modifier for making class variables and methods.

- Final

  The final modifier for concluding the executions of classes, variables and methods.

- Abstract

  This modifier is used for for creating abstract methods and classes.

- Volatile and Synchronized

  These modifiers are typically used for threads.

Operators in Java

The operator set in Java is extremely rich. Broadly, the operators available in Java are divided in the following categories.

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Bitwise Operators
- Misc Operators
- Assignment Operators

The Arithmetic Operators

Operations in Java are used in essentially the same manner as in algebra. They are used with variables for performing arithmetic operations. Here is a list of arithmetic operators available in Java.

Operation	Operator	Description
Addition	+	Adds the values of two variables
Subtraction	-	Subtracts the values of two variables
Multiplication	*	Multiplies the values of two variables
Division	/	Divides the values of two variables
Modulus	%	The resultant value is the the remainder of division
Increment	++	Increases the value by 1

Decrement	—	Decreases the value by 1

The Relational Operators

Java also supports several relational operators. The list of relational operators that are supported by Java are given below.

Operation	Operator	Description
Equal To	==	Compares the values of two variables for equality
Not Equal To	!=	Compares the values of two variables for inequality
Greater Than	>	Checks if one value is greater than the other value
Lesser Than	<	Checks if one value is lesser than the other value
Greater Than Or Equal To	>=	Checks if one value is greater than or equal to the other value
Lesser Than Or Equal To	<=	Checks if one value is lesser than or equal to the other value

The Bitwise Operators

The bitwise operators available in Java can be easily applied to a number of data types. These data types include byte, short, long, int and char. Typically, any bitwise operator performs the concerned operation bit-wise. For instance, if you consider the example of an integer x, which has the value 60. Therefore, the binary equivalent of x is 00111100. Consider another variable y, with the value 13 or 00001101. If we perform the bitwise

operation & on these two numbers, then you will get the following result:

x&y = 0000 1100

The table shown below shows a list of bitwise operators that are available in Java.

Operation	Operator	Description
BINARY AND	&	Performs the AND operation
BINARY OR	|	Performs the OR operation
BINARY XOR	^	Performs the XOR operation
ONE'S COMPLEMENT	~	Performs the complementation operation on a unary variable
BINARY LEFT SHIFT	<<	Performs the left shifting of bits
BINARY RIGHT SHIFT	>>	Performs the right shifting of bits

In addition to the above mentioned, Java also supports right shift zero fill operator (>>>), which fills the shifted bits on the right with zero.

The Logical Operators

Logical operators are an integral part of any operator set. The logical operators supported by Java are listed in the table below.

Operation	Operator	Description

Logical AND	&&	Returns True if both the conditions mentioned are true
Logical OR	||	Returns True if one or both the conditions mentioned are true
Logical NOT	!	Returns True if the condition mentioned is False

The Assignment Operators

There are following assignment operators supported by Java language:

Operation	Operator	Description
Simple assignment operator	=	Assigns a value on the right to the variable in the left
Add - assignment operator	+=	Adds the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left
Subtract - assignment operator	-=	Subtracts the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left
Multiply - assignment operator	*=	Multiplies the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left

Divide - assignment operator	/=	Divides the value on the right to the value of the variable on the left and assigns the resultant to the variable on the left
Modulus - assignment operator	%=	It takes the modulus of the LHS and RHS and assigns the resultant to the variable on the left
Left shift - assignment operator	<<=	It takes the left shift of the LHS and RHS and assigns the resultant to the variable on the left
Right shift - assignment operator	>>=	It takes the right shift of the LHS and RHS and assigns the resultant to the variable on the left
Bitwise - assignment operator	&=	It takes the bitwise AND of the LHS and RHS and assigns the resultant to the variable on the left
bitwise exclusive OR - assignment operator	^=	It takes the bitwise XOR of the LHS and RHS and assigns the resultant to the variable on the left
bitwise inclusive OR - assignment operator	|=	It takes the bitwise OR of the LHS and RHS and assigns the resultant to the variable on the left

Misc Operators

In addition to the above mentioned, there are several other operators, which are supported by Java.

Conditional Operator (? :):

The conditional operator is a ternary operator that contains three operands. Essentially, this operator is used for the evaluation of boolean expressions. The operator tests the first operand or condition and if the condition is true, then the second value is assigned to the variable. However, if the condition is false, the third operand is assigned to the variable. The syntax of this operator is as follows:

variable a = (<condition>) ? valueiftrue : valueiffalse

Sample implementation:

```
public class myTest {

public static void main(String args[]){

int x, y;

x = 5;

y = (x == 5) ? 15: 40;

System.out.println( “y = ” +  y );

y = (x == 34) ? 60: 95;

System.out.println( “x = ” + y );

}

}
```

The compilation and execution of this code shall give the following result:

y = 15

y = 95

instanceof Operator:

Only object reference variables can be used with this operator. The objective of this operator is to check is an object is an instance of an exiting class. The syntax of this operator is as follows:

(<object reference variable>) instanceof(<interface/class>)

Sample implementation of this operator and its purpose of use is given below:

```
public class myTest {

public static void main(String args[]){

int x = 4;

boolean resultant = x instanceof int;

System.out.println( resultant );

}

}
```

The output of this code shall be true. This operator can also be used in comparison. A sample implementation of this is given below:

```
class Animal {}

public class Monkey extends Animal {

public static void main(String args[]){

Animal newa = new Monkey();

boolean resultant =  newa instanceof Monkey;

System.out.println( resultant );

}

}
```

The output for this code will also be true.

Precedence of Java Operators

More often than not, operators are used in combinations in expressions. However, you must have also realized that it becomes difficult to predict the order in which operations will take place during execution. The operator precedence table for Java shall help you predict operator operations in an expression deduction. For instance, if you are performing addition and multiplication in the same expression, then multiplication takes place prior to addition. The following table illustrates the order and hierarchy of operators in Java. The associativity for all the operators is left to right. However, the unary, assignment and conditional operator follows right to left associativity.

Operator	**Category**
() [] . (dot operator)	Postfix
++ - - ! ~	Unary
* / %	Multiplicative
+ -	Additive
>> >>> <<	Shift
> >= < <=	Relational
== !=	Equality
&	Bitwise AND

|	Bitwise OR
^	Bitwise XOR
&&	Logical AND
||	Logical OR
?:	Conditional
= += -= *= /= %= >>= <<= &= ^= |=	Assignment
,	Comma

Loops in Java

Looping is a common programming situation that you can expect to encounter rather regularly. Loop can simply be described as a situation in which you may need to execute the same block of code over and over. Java supports three looping constructs, which are as follows:

- for Loop
- do…while Loop
- while Loop

In addition to this, the foreach looping construct also exists. However, this construct will be explained in the chapter on arrays.

The while Loop:

A while loop is a control structure that permits you to rehash an errand a specific number of times. The syntax for this construct is as follows:

while(boolean_expression) {

/Statements

}

At the point when executing, if the boolean_expression result is genuine, then the activities inside the circle will be executed. This will proceed till the time the result for the condition is genuine. Here, key purpose of the while loop is that the circle may not ever run. At the point when the interpretation is tried and the result is false, the body of the loop will be skipped and the first proclamation after the while circle will be executed.

Sample:

```
public class myTest {

public static void main(string args[]) {

int i=5;

while(i<10) {

System.out.print(” i = ” + i );

i++;

System.out.print(“\n”);

}

}
```

This would deliver the accompanying result:

```
x = 5

x = 6

x = 7

x = 8

x = 9

x = 5
```

The do…while Loop

A do…while loop is similar to the while looping construct aside from that a do…while circle is ensured to execute no less than one time. The syntax for this looping construct is as follows:

```
do {

/Statements
```

}while(<booleanexpression>);

Perceive that the Boolean declaration shows up toward the end of the circle, so the code execute once before the Boolean is tried. In the event that the Boolean declaration is genuine, the stream of control bounced go down to do, and the code execute once more. This methodology rehashes until the Boolean articulation is false.

Sample implementation:

```
public class myTest {

public static void main(string args[]){

int i = 1;

do{

System.out.print(“i = ” + i );

i++;  System.out.print(“\n”);

}while( i<1 );

}
```

This would create the accompanying result:

```
i = 1
```

The for Loop

A for circle is a reiteration control structure that permits you to effectively compose a loop that needs to execute a particular number of times. A for looping construct is helpful when you know how often an errand is to be rehashed. The syntax for the looping construct is as follows:

The punctuation of a for circle is:

for(initialization; Boolean_expression; redesign)

{

/Statements

}

Here is the stream of control in a for circle:

- The introduction step is executed in the first place, and just once. This step permits you to pronounce and introduce any loop control variables. You are not needed to put an announcement here, the length of a semicolon shows up.
- Next, the Boolean outflow is assessed. In the event that it is genuine, the assemblage of the loop is executed. In the event that it is false, the assortment of the loop does not execute and stream of control hops to the following articulation past the for circle.
- After the group of the for circle executes, the stream of control bounced down to the overhaul explanation. This announcement permits you to overhaul any circle control variables. This announcement can be left clear, the length of a semicolon shows up after the Boolean declaration.
- The Boolean outflow is currently assessed once more. On the off chance that it is genuine, the loop executes and the scope rehashes itself. After the Boolean declaration is false, the for loop ends.

Sample Implementation

public class myTest {

public static void main(string args[]) {

for(int i = 0; i < 5; i = i+1) {

System.out.print(“i = ” + i);

System.out.print(“\n”);

}

}

This would deliver the accompanying result:

i = 0

i = 1

i = 2

i = 3

i = 4

Extended Version of for Loop in Java

As of Java 5, the upgraded for loop was presented. This is basically utilized for Arrays. The syntax for this loop is as follows:

for(declaration : statement)

{

//Statements

}

- Declaration: The recently declared variable, which is of a sort perfect with the components of the show you are getting to. The variable will be accessible inside the for piece and its esteem would be the same as the current array component.
- Expression: This assesses to the exhibit you have to loop through. The interpretation can be an array variable or function call that returns an array.

Sample Implementation:

```
public class myTest {

public static void main(string args[]){

int [] mynumber = {0, 5, 10, 15, 20};

for(int i : mynumber ){

System.out.print( i );

System.out.print(” ,”);

}

}
```

This would deliver the accompanying result:

0, 5, 10, 15, 20

The break Keyword

The break keyword is utilized to stop the whole loop execution. The break word must be utilized inside any loop or a switch construct. The break keyword will stop the execution of the deepest circle and begin executing the following line of code after the ending curly bracket. The syntax for using this keyword is as follows:

```
break;
```

Sample Implementation:

```
public class myTest {

public static void main(string args[]) {

int [] mynumbers = {0, 5, 10, 15, 20};

for(int i : mynumbers ) {

if( i == 15 ) {
```

```
break;
}
System.out.print( i );
System.out.print("\n");
}
}
```

This would deliver the accompanying result:

```
0
5
10
```

The Continue Keyword

The proceed with decisive word can be utilized as a part of any of the loop control structures. It causes the loop to quickly bounce to the following emphasis of the loop.

- In a for circle, the continue keyword reasons stream of control to quickly bounce to the overhaul articulation.
- In a while or do/while loop, stream of control instantly hops to the Boolean interpretation.

The syntax of using this keyword is as follows:

```
continue;
```

Sample Implementation:

```
public class myTest {
public static void main(String args[]) {
```

```
int [] mynumbers = {0, 5, 10, 15, 20};

for(int i : mynumbers ) {

if( i == 15 ) {

continue;

}

System.out.print( i );

System.out.print(“\n”);

}

}

}
```

The expected output of the code is:

0

5

10

20

Decision Making

There are two sorts of decision making constructs in Java. They are:

- if constructs
- switch constructs

The if Statement:

An if constructs comprises of a Boolean outflow emulated by one or more proclamations. The syntax for using this construct is as follows:

if(<condition>) {

//Statements if the condition is true

}

In the event that the Boolean construct assesses to true, then the scope of code inside the if proclamation will be executed. If not the first set of code after the end of the if construct (after the end wavy prop) will be executed.

Sample Implementation:

public class myTest {

public static void main(string args[]){

int i = 0;

if(i < 1){

System.out.print(“The if construct is executing!”);

}

}

This would create the accompanying result:

The if construct is executing!

The if…else Statement

An if proclamation can be trailed by a non-compulsory else explanation, which executes when the Boolean outflow is false. The syntax for this construct is as follows:

if(<condition>){

//Executes if condition is true

}

else{

//Executes if condition is false

}

Sample Implementation:

public class myTest {

public static void main(string args[]){

int i = 0;

if(i > 1){

System.out.print(“The if construct is executing!”);

}

else{

System.out.print(“The else construct is executing!”);

}

}

This would create the accompanying result:

The else construct is executing!

The if…else if Statement

An if proclamation can be trailed by a non-compulsory else if…else explanation, which is exceptionally helpful to test different conditions utilizing single if…else if articulation.

At the point when utilizing if , else if , else proclamations there are few focuses to remember.

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to numerous else if's and they must precede the else.
- If one of the if conditions yield a true, the other else ifs and else are ignored.

The syntax for using this decision making construct Is as follows:

```
if(condition_1){

//Execute if condition_1 is true

}

else if(condition_2){

//Execute if condition_2 is true

}

else if(condition_3){

//Execute if condition_3 is true

}

else
```

{

//Execute if all conditions are false

}

Sample Implementation:

public class myTest {

public static void main(string args[]){

int i = 0;

if(i > 1){

System.out.print(“The first if construct is executing!”);

}

else if(i == 0){

System.out.print(“The second if construct is executing!”);

}

else{

System.out.print(“The else construct is executing!”);

}

}

This would create the accompanying result:

The second if construct is executing!

Nested if…else Statement

It is legitimate to home if-else constructs, which implies you can utilize one if or else if

proclamation inside an alternate if or else if explanation. The syntax for using this construct Is as follows:

```
if(condition_1){
//Execute if condition_1 is true
        if(condition_2){
        //Execute if condition_2 is true
        }
}
else if(condition_3){
//Execute if condition_3 is true
}
else
{
//Execute if all conditions are false
}
```

Sample Implementation:

```
public class myTest {
public static void main(string args[]){
int i = 1;
if( i >= 1 ){
System.out.println(“The if construct is executing!”);
```

```
        if(i == 1){
        System.out.println(“The nested if construct is executing!”);
        }
}
else{
System.out.print(“The else construct is executing!”);
}
}
```

This would create the accompanying result:

The if construct is executing!

The nested if construct is executing!

The switch Statement

A switch construct permits a variable to be tried for equity against a rundown of values. Each one value is known as a case, and the variable being exchanged on is checked for each one case. The syntax for using this decision making construct is as follows:

```
switch(<condition>){
case value1:
//Statements
break;
case value2 :
//Statements
break;
```

default:

//Optional

}

The accompanying runs apply to a switch construct:

- The variable utilized as a part of a switch explanation must be a short, byte, char or int.
- You can have any number of case explanations inside a switch. Each one case is trailed by the value to be contrasted with and a colon.
- The value for a case must be the same type as the variable in the switch and it must be a steady or an exacting value.
- When the variable being exchanged on is equivalent to a case, the announcements after that case will execute until a break is arrived at.
- When a break is arrived at, the switch ends, and the stream of control bounces to the following line after the switch.
- Not each case needs to contain a break. In the event that no break shows up, the stream of control will fall through to consequent cases until a break is arrived at.
- A switch articulation can have a discretionary default case, which must show up toward the end of the switch. The default case can be utilized for performing an undertaking when none of the cases is true. No break is required in the default case. However, as per the convention, the use of the same is recommended.

Sample Implementation:

public class myTest {

public static void main(string args[]){

char mygrade = ‘A’;

```
switch(mygrade)

{

case “A” :

System.out.println(“Excellent Performance!”);

break;

case “B” :

System.out.println(“Good Performance!”);

break;

default :

System.out.println(“Failed”);

}
```

Aggregate and run above code utilizing different inputs to grade. This would create the accompanying result for the present value of mygrade:

Excellent Performance!

Strings in Java

Strings, which are generally utilized as a part of Java, for writing computer programs, are a grouping of characters. In the Java programming language, strings are like everything else, objects. The Java platform provides the String class to make and control strings.

Instantiating Strings

The most appropriate approach to make a string is to use the following statement:

String mystring = “Hi world!”;

At whatever point it experiences a string exacting in your code, the compiler makes a String object with its value for this situation, “Hi world!’.

Similarly as with other objects, you can make Strings by utilizing a constructor and a new keyword. The String class has eleven constructors that permit you to give the starting estimation of the string utilizing diverse sources, for example, a cluster of characters.

public class myStringdemo{

public static void main(string args[]){

char[] myarray = { ‘h’, ‘i’, ‘.’};

String mystring = new String(myarray);

System.out.println(mystring);

}

This would deliver the accompanying result:

hi.

Note: The String class is changeless, so that once it is made a String object, its type can’t be changed. In the event that there is a need to make a great deal of alterations to Strings

of characters, then you ought to utilize String Buffer & String Builder Classes.

Determining String Length

Routines used to get data about an object are known as accessor methods. One accessor technique that you can use with strings is the length() function, which furnishes a proportional payback of characters contained in the string item. This function can be utilized in the following manner:

public class mystring {

public static void main(string args[]) {

String newstr = “I am hungry!”;

int strlen = newstr.length();

System.out.println(“length = ” + strlen); }

This would create the accompanying result:

length = 12

How to Concatenate Strings

The String class incorporates a function for connecting two strings:

mystring1.concat(mystring2);

This returns another string that is mystring1 with mystring2 added to it toward the end. You can likewise utilize the concat() system with string literals, as in:

“My name is “.concat(“mary”);

Strings are all the more usually concatenated with the + administrator, as in:

“Hi,” + ” world” + “!”

which brings about:

“Hi, world!”

Sample Implementation:

```
public class MyString {

public static void main(string args[]) {

String mystr = “Sorry”;

System.out.println(“I ” + “am ” + mystr);

}
```

This would deliver the accompanying result:

I am Sorry

Format Strings

You have format() and printf() functions to print the output with designed numbers. The function format() of the String class returns a String object as against a Printstream object. This function creates a formatted string that can be reused. This function can be used in the following manner:

```
String fstr;

fstr = String.format(“Float variable value ” + “%f, and Integer value ” +  “variable is %d, and the contents of the string is ” + ” %s”, fVar, iVar, sVar);

System.out.println(fstr);
```

String Methods

This section contains a list of methods that are available as part of the String class.

int compareTo(Object obj) – This function compares the specified string with the object concerned.

char charAt(int chindex) – This function returns the char present at the index value ‘index.’

int compareToIgnoreCase(String mystr) – This function performs the lexographic comparison of the two strings. However, the case differences are ignored by this function.

int compareTo(String aString) – This function performs the lexographic comparison between the strings.

boolean contentEquals(StringBuffer strb) – This function checks if the string is same as the sequence of characters present in the StringBuffer. It returns true on success and false on failure.

String concat(String strnext) – This function appends the string with another string at the end.

static String copyValueOf(char[] mydata, int xoffset, int xcount) – This function returns a stringf, which is indicative of the character sequence in the original string.

static String copyValueOf(char[] newdata) – This function copies the string of characters into a character buffer in the form of a sequence of characters.

boolean equals(Object aObject) – This function compares the object with the string concerned.

boolean endsWith(String newsuffix) – This function appends the string with the specified suffix.

byte getBytes() – Using this function, the string can be encoded into bytes format, which are stored in a resultant array.

boolean equalsIgnoreCase(String aString) – This function makes a comparison of the two strings without taking the case of characters into consideration.

void getChars(int srcBegin, int sourceEnd, char[] dst, int destinationBegin) – This

function copies characters from the specified beginning character to the end character into an array.

byte[] getBytes(String charsetnm) - Using this function, the string can be encoded into bytes format using named char set, which are stored in a resultant array.

int indexOf(int charx) – This function returns the index of first character that is same as the character specified in the function call.

int hashCode() – A hash code is returned by this string.

int indexOf(String newstr) - This function returns the index of the first occurrence of a substring in a string.

int indexOf(int charx, int fromIndexloc) – This function returns the index of first character that is same as the character specified in the function call. The search starts from the specified index.

String intern() – A canonical representation of a string object given in the function call is returned.

int indexOf(String newstr, int fromIndexloc) - This function returns the index of the first occurrence of a substring in a string. The search starts from the specified index.

int lastIndexOf(int charx, int fromIndexloc) - This function makes a search for a character from the specified index and returns the index where the last occurrence is found.

int lastIndexOf(int charx) - This function makes a search for a character backwards and returns the index where the last occurrence or first occurrence in a backward search is found.

int lastIndexOf(String newstr, int fromIndexloc) – This function makes a search for a sub-string from the specified index and returns the index where the last occurrence is found.

int lastIndexOf(String newstr) - This function makes a search for a sub-string backwards

and returns the index where the last occurrence or first occurrence in a backward search is found.

boolean matches(String aregex) - – This function checks for equality between a string region and a regular expression.

int length() – This function calculates and returns the string length.

boolean regionMatches(int totaloffset, String otherstr, int otheroffset, int strlen) – This function checks for equality between string regions.

boolean regionMatches(boolean ignorecharcase, int totaloffset, String otherstr, int otheroffset, int strlen) – This function checks for equality between string regions.

String replace(char oldCharx, char newCharx) - This function looks for a substring that matches the regular expression and then replaces all the occurrences with the specified string. The function returns the resultant string, which is obtained after making all the replacements.

String replaceFirst(String newregex, String newreplacement) – This function looks for a substring that matches the regular expression and then replaces the first occurrence with the specified string.

String replaceAll(String newregex, String xreplacement) - This function looks for a substring that matches the regular expression and then replaces all the occurrences with the specified string.

String[] split(String newregex, int xlimit) – This function performs splitting of the string according to the regular expression with it and the given limit.

String[] split(String newregex) - This function performs splitting of the string according to the regular expression with it.

boolean startsWith(String newprefix, int totaloffset) – This function checks if the given

string has the prefix at the specified index.

boolean startsWith(String newprefix) – This function checks if the given string begins with the prefix sent with the function call.

String substring(int beginIndexloc) - This function returns a string, which is substring of the specified string. The substring is determined by the beginning index to the end of the string.

CharSequence subSequence(int beginIndexloc, int endIndexloc) - This function returns a character sequence, which is sub-character sequence of the specified character sequence. The substring is determined by the beginning and ending indexes.

char[] toCharArray() – This function performs the conversion of a string to a character array.

String substring(int beginIndexloc, int endIndexloc) – This function returns a string, which is substring of the specified string. The substring is determined by the beginning and ending index.

String toLowerCase(Locale localenew) - This function converts all the characters in the specified string to lower case using given locale rules.

String toLowerCase() - This function converts all the characters in the specified string to lower case using default locale rules.

String toUpperCase() - This function converts all the characters in the specified string to upper case using default locale rules.

String toString() – This function returns the string itself.

String toUpperCase(Locale localenew) – This function converts all the characters in the specified string to upper case using locale rules.

static String valueOf(primitive data type x) – A string representation is returned by this

function.

String trim() – Omits the whitespace that trails and leads a string.

Arrays

Java supports an information structure, which is similar to a cluster. This information structure is called an array. It is capable of storing an altered size successive accumulation of components of the same data type. An array is utilized to store an accumulation of information, yet it is frequently more valuable to think about it as an exhibit for storing variables of the same sort.

As opposed to making declarations of individual variables, for example, num0, num1 and num99, you can declare one array variable. For example, an array of four elements is declared as arrayname[4]. This chapter discusses all the facets of array declaration, access and manipulation.

How To Declare array Variables

To utilize an array as a part of a system, you must declare a variable to reference the array. Besides this, you must determine the sort of array the variable can reference. Here is the syntax for declaring a variable of the type array:

datatype[] myarray;

Sample Implementation:

The accompanying code bits are illustrations of this concept:

double[] myarray;

Making Arrays

You can make an exhibit by utilizing the new operator with the accompanying statement:

myarray = new datatype[sizeofarray];

The above declaration does two things:

- It makes an exhibit with the help of the new operator in the following manner:

  new datatype[arraysize];

- It relegates the reference of the recently made array to the variable myarray.

Proclaiming a array variable, making an exhibit, and doling out the reference of the show to the variable can be consolidated in one declaration, as appeared:

datatype[] myarray = new datatype[sizeofarray];

On the other hand, you can also make clusters in the following manner:

datatype[] myarray = {val0, val1, …, valk};

The components of the array are gotten to through the record. Array lists are 0-based; that is, they begin from 0 to go up to myarray.length-1.

Sample Implementation:

The declaration shown below declares an array, myarray, makes a cluster of 10 components of double type and doles out its reference to myarray:

double[] myarray = new double[10];

Handling Arrays

At the point when handling components of an array, we frequently utilize either for or foreach in light of the fact that the majority of the components in an array are of the same sort and the extent of the exhibit is known.

Example:

public class Mytestarray {

public static void main(string[] args) {

double[] myarray = {0.5, 1.2, 2.2, 3.4, 4.7};

```
for (int k = 0; k < myarray.length; k++) {

System.out.println(myarray[k] + ” “);

}

double aggregate = 0;

for (int k = 0; k < myarray.length; k++) {

aggregate += myarray[k];

}

System.out.println(“Aggregate value = ” + aggregate);

double maxval = myarray[0];

for (int k = 1; k < mylist.length; k++) {

if (myarray[i] > maxval)

maxval = myarray[k];

}

System.out.println(“Max Value is ” + maxval);

}
```

This would create the accompanying result:

0.5 1.2 2.2 3.4 4.7

Aggregate = 12.0

Max Value is 4.7

The foreach Loops

JDK 1.5 presented another for construct, which is known as foreach loop or extended for

loop. This construct empowers you to cross the complete array successively without utilizing an extra variable.

Sample Implementation:

```
public class Mytestarray {

public static void main(string[] args) {

double[] myarray = {0.5, 1.2, 2.2, 3.4, 4.7};

for (double i: myarray) {

System.out.println(i);

}

}
```

This would deliver the accompanying result:

0.5 1.2 2.2 3.4 4.7

Passing Arrays to Methods:

Generally, just as you can pass primitive values to methods or functions, you can likewise pass arrays to systems. Case in point, the accompanying method shows the components in an int array:

```
public static void printarr(int[] arr) {

for (int k = 0; k < arr.length; k++) {

System.out.print(arr[k] + ” “);

}
```

You can summon it by passing an array. Case in point, the accompanying declaration conjures the printarr function to show the elements of the array.

printarr(new int[]{0, 3, 5, 3, 1});

The compilation and execution of this code yields the following result:

0 3 5 3 1

How Can A Method Return An Array

A system might likewise give back an array. Case in point, the method demonstrated underneath returns an array that is the inversion of an alternate array:

public static int[] revarr(int[] myarr) {

int[] resultarr = new int[myarr.length];

for (int k = 0, i = resultarr.length - 1; k <= myarr.length/2; k++, i- -) {

resultarr[j] = myarr[k];

}

return resultarr;

}

The Arrays Class

The java.util.arrays class contains different functions for sorting and seeking values from array, looking at arrays, and filling components into arrays. These functions are available for all primitive data types.

- public static boolean equals(long[] a, long[] a2) - returns true if the two indicated arrays are equivalent to each other. Two arrays are viewed as equivalent if both of them contain the same number of components, and all relating sets of components in the two arrays are equivalent. This returns true if the two shows are equivalent. Same function could be utilized by all other primitive data types.
- public static int binarysearch(object[] an, Object key) - looks the pointed out array

of Object for the defined value utilizing the double calculation. The array must be sorted before making this call. This returns list of the keys, in the event that it is contained in the list; generally, (-(insertion point + 1).

- public static void sort(Object[] a) – This function can be used to sort a given array in the ascending order. It can likewise be used for any data type.
- public static void fill(int[] an, int val) - appoints the detailed int value to every component of the pointed out array of ints. Same function could be utilized for arrays of other data types as well.

Regular Expressions

Java includes the java.util.regex package to match with regular expressions. Java's normal outflows are fundamentally the same to the Perl programming language and simple to learn. A consistent outflow is an exceptional succession of characters that helps you match or discover different strings or sets of strings, utilizing a specific syntax held as a part of an example. They can be utilized to find, alter, or control content and information. The java.util.regex package essentially comprises of the accompanying three classes:

- Pattern Class: A Pattern article is an arranged representation of a consistent declaration. The Pattern class does not have any public constructors. To make an example, you should first conjure one of its public static methods, which will then give back a Pattern object. These functions acknowledge a normal statement as the first contention.
- Matcher Class: A Matcher article is the motor that translates the example and performs match operations against an information string. Like the Pattern class, Matcher has no public constructors. You get a Matcher object by conjuring the matcher method on a Pattern object.
- Patternsyntaxexception: A Patternsyntaxexception object is an unchecked exemption that shows a sentence structure mistake in a consistent statement design.

Catching Groups

Catching groups are an approach to treat various characters as an issue unit. They are made by putting the characters to be assembled inside a set of enclosures. Case in point, the normal declaration (canine) makes a solitary gathering containing the letters "d", "o", and "g". Catching gatherings are numbered by numbering their opening enclosures from left to right. In the representation ((A)(b(c))), for instance, there are four such gatherings:

- (a)
- (c)
- (b(c))
- ((a)(b(c)))

To discover what number of gatherings are available in the declaration, call the groupcount strategy on a matcher object. The groupcount technique gives back an int demonstrating the quantity of catching gatherings show in the matcher's example. There is likewise an uncommon gathering, gathering 0, which dependably speaks to the whole outflow. This gathering is excluded in the aggregate reported by groupcount.

Sample Implementation:

This sample code emulates how to discover from the given alphanumeric string a digit string:

```
import java.util.regex.matcher;

import java.util.regex.pattern;

public class Myregexmatches {

public static void primary( String args[] ){

String line = "Request for Qt3000! ";

String example = "(.*)(\d+)(.*)";

Pattern myr = Pattern.compile(pattern);

Matcher mym = myr.matcher(line);

if (mym.find( )) {

System.out.println("Value = " + mym.group(0) );

System.out.println("Value = " + mym.group(1) );
```

```
System.out.println(“Value = ” + mym.group(2) );

}

else {

System.out.print(“No match found!”);

}

}
```

Regular Expression Syntax

Given below is a list of regular expression syntax for your reference.

Matches	**Subexpression**
Matches line beginning	^
Matches line end	$
Matches single characters except for the newline character	.
Matches single character in braces	[…]
Matches single character, which are not in braces	[^…]
String beginning	\A
String end	\z
String end except for the final line terminating character	\Z
Matches 0 or more instances of expression	re*

Matches 1 or more instances of the expression	re+
Matches 0 or 1 instances of expression.	re?
Matches exactly n of instances of expression.	re{ n}
Matches n or more instances of the specified expression.	re{ n,}
Matches minimum n and maximum m instances of the expression.	re{ n, m}
Matches one of these: a or b.	a| b
Groups regular expressions. The matching text is remembered.	(re)
Groups regular expressions. The text is not remembered.	(?: re)
Matches independent pattern. No backtracking is supported.	(?> re)
Matches characters in a word.	\w
Matches characters, which are non-word.	\W
Matches whitespace. These characters are equivalent to [\t\n\r\f].	\s
Matches space, which is non-whitespace.	\S
Matches digits. These are typically equal to 0 to 9.	\d
Matches non-digits.	\D
Matches string beginning.	\A

Matches string end just before the newline character appears.	\Z
Matches string end.	\z
Matches the point where the last matching condition was found.	\G
Group number n back-reference	\n
Matches boundaries of the word when used without brackets. However, backspace is matched when it is used inside brackets.	\b
Matches boundaries, which are non-word	\B
Matches carriage returns, newlines, and tabs	\n, \t, etc.
Escape all the characters until a \E is found	\Q
Ends any quotes that begin with \Q	\E

Methods of the Matcher Class

Index Methods:

The following table gives a list of methods hat show correctly where the match was found in the info string:

- public int start(int bunch)

  Furnishes a proportional payback record of the subsequent caught by the given group amid the past match operation.

- public int begin()

  Furnishes a proportional payback record of the past match.

- public int end(int bunch)

  Furnishes a proportional payback after the last character of the subsequent caught by the given group amid the past match operation.

- public int end()

  Furnishes a proportional payback after the last character matched.

Study Methods:

Study methods survey the info string and return a Boolean demonstrating whether the example is found:

- public boolean find()

  Endeavors to discover the following subsequence of the info arrangement that matches the example.

- public boolean lookingat()

  Endeavors to match the info arrangement, beginning toward the start of the district, against the example.

- public boolean matches()

  Endeavors to match the whole district against the example.

- public boolean find(int begin)

  Resets this matcher and after that endeavors to discover the following subsequence of the information grouping that matches the example, beginning at the detailed list.

Substitution Methods:

Substitution methods are valuable methods for supplanting content in a data string:

- public static String quotereplacement(string mystr)

  Gives back an exacting substitution String for the tagged String. This method creates a String that will function as an issue substitution s in the appendreplacement system for the Matcher class.

- public Stringbuffer appendtail(stringbuffer strbuff)

  Actualizes a terminal annex and-supplant step.

- public Matcher appendreplacement(stringbuffer strbuff, String strsubstitution)

  Actualizes a non-terminal annex and-supplant step.

- public String replacefirst(string strsubstitution)

  Replaces the first subsequence of the data succession that matches the example with the given substitution string.

- public String replaceall(string strsubstitution)

  Replaces each subsequence of the data succession that matches the example with the given substitution string.

The begin and end Methods:

Taking after is the sample that tallies the quantity of times the statement “felines” shows up in the data string:

```
import java.util.regex.pattern;

import java.util.regex.matcher;

public class Regexmatches {

private static last String INPUT =  “feline cattie feline”;

private static last String REGEX = “\bcat\b”;
```

```
public static void principle( String args[] ){

Pattern myp = Pattern.compile(regex);

Matcher mym = myp.matcher(input);

int checkval = 0;

while(mym.find()) {

count++;

System.out.println(“match number “+count);

System.out.println(“start(): “+mym.start());

System.out.println(“end(): “+mym.end());

}

}
```

You can see that this sample uses word limits to guarantee that the letters “c” “a” “t” are not only a substring in a more extended word. It likewise provides for some helpful data about where in the information string the match has happened. The begin technique gives back where its due record of the subsequence caught by the given group amid the past match operation, and end furnishes a proportional payback of the last character matched, in addition to one.

The matches and lookingat Methods:

The matches and lookingat methods both endeavor to match an information succession against an example. The distinction, then again, is that matches requires the whole enter grouping to be matched, while lookingat does not. Both techniques dependably begin toward the start of the data string.

The replacefirst and replaceall Methods:

The replacefirst and replaceall routines supplant content that matches a given standard representation. As their names show, replacefirst replaces the first event, and replaceall replaces all events.

The appendreplacement and appendtail Methods:

The Matcher class additionally gives appendreplacement and appendtail routines to content substitution.

PatternSyntaxException Class Methods:

A PatternSyntaxException is an exception, which is unchecked. This exception indicates a syntactical error in the pattern of the regular expression. The PatternSyntaxException class offers the following methods to the developer for use.

- public int getIndex()

  This function returns the index of error.

- public String getDescription()

  This function returns the description of error.

- public String getMessage()

  This function returns the description and index of error.

- public String getPattern()

  This function returns the error-causing regular expression pattern.

Methods

A Java method is an accumulation of explanations that are gathered together to perform an operation. When you call the System.out.println function, for instance, the framework executes a few articulations so as to show a message on the output screen. Presently, you will figure out how to make your own routines with or without return qualities, call a method with or without parameters, over-loaded methods utilizing the same names, and apply method deliberation in the system plan.

How To Create Methods

Considering the accompanying sample to clarify the structure of a method:

```
public static int functionname(int x, int y) {
//Statements
}
```

Here, the method uses the following elements:

- Modifier: public static
- Data type of the return value: int
- Method name: functionname
- Formal Parameters: x, y

Such constructs are otherwise called Functions or Procedures. However, there is a distinctive quality of these two:

- Functions: They return an explicit value.
- Procedures: They don’t give back any quality.

Function definition comprises of a system header and body. The construct given above can

be generalized to the following arrangement:

modifier returndatatype methodname (List of Parameter) { //Statements }

The structure indicated above incorporates:

- Modifier: It characterizes the right to gain entrance to the method and it is non-compulsory to utilize.
- Returntype: Method may give back a value of this data type.
- Methodname: This is the method name, which comprise of the name and the parameter list of the method.
- List of Parameters: The rundown of parameters, which entails data type, request, and number of parameters of a method. A method may contain zero parameters as well.
- Statements: The method body characterizes what the method does with explanations.

Sample Implementation:

Here is the source code of the above characterized method called maxval(). This technique takes two parameters number1 and number2 and furnishes a proportional payback between the two:

```
public static int minval(int num1, int num2) {

int minvalue;

if (num1 > num2)  minvalue = num2;

else  minvalue = num1;

return minvalue;

}
```

Calling A Method

For utilizing a method, it ought to be called. There are two courses in which a technique is called i.e. technique gives back a value or nothing (no return value). The methodology of system calling is basic. At the point when a project summons a method, the system control gets exchanged to the called method. This called method then returns control to the guest in two conditions. These conditions include:

- Reaches the method closure brace.
- Return articulation is executed.

The methods returning void is considered as call to an announcement. Lets consider a sample:

System.out.println(“This is the end of the method!”);

The method returning a value can be seen by the accompanying illustration:

double resultant = sumval(4.2, 2.5);

Sample Implementation:

public class Minnumber{

public static void main(string[] args) {

double x = 21.5;

double y = 2.0;

double z = minvalfunc (x, y);

System.out.println(“The returned value = ” + z);

}

public static double minvalfunc (double num1, double num2) {

```
double minval;

if (num1 > num2)

minval = num2;

else

minval = num1;

return minval;

}
```

This would create the accompanying result:

The returned value = 2.0

The void Keyword:

The void keyword permits us to make methods, which don’t give back a value. Here, in the accompanying illustration we’re considering a void method. This function is a void method, which does not give back any value. Call to a void system must be an announcement i.e. rankpoints(657.3);. It is a Java explanation which closes with a semicolon as appeared.

Sample Implementation:

```
public class Myexample {

public static void main(string[] args) {

rankpoints(657.3);

}

public static void rankpoints(double valfoc) {

if (valfoc >= 100.5) {
```

```
System.out.println(“A1 Rank”);

}

else if (valfoc >= 55.4) {

System.out.println(“A2 Rank”);

}

else {

System.out.println(“A3 Rank”);

}

}
```

This would deliver the accompanying result:

A1 Rank

Passing Parameters by Value

While working under calling procedure, contentions is to be passed. These ought to be in the same request as their particular parameters in the function call. Parameters can be passed by reference or value. Passing parameters by value means calling a method with a parameter. Through, this is the contention value is gone to the parameter.

Sample Implementation:

The accompanying project demonstrates an illustration of passing parameter by value.

```
public class Myswapping {

public static void main(string[] args) {

double x = 0.43;

double y = 34.65;
```

```
System.out.println(“Values of X and Y before swapping, x = ” + x + ” and y = ” + y);

myswapfunc (x, y);

System.out.println(“\nValues of X and Y after swapping: “);

System.out.println(“x = ” +  x + ” and y is ” + y);

}

public static void myswapfunc (int x, int y) {

System.out.println(“Inside the function: Values of X and Y before swapping, x = ” + x  + ”
y = ” + y);

System.out.println(“Inside the function: Values of X and Y after swapping, x = ” + x  + ” y
= ” + y);

}
```

TO READ MORE, SEARCH AMAZON FOR: “Java For Beginners

A Simple Start To Java Programming Sanderson”